

A Survey of the Use of Test Report in Crowdsourced Testing

Song Huang, Hao Chen*, Zhanwei Hui*, Yuchan Liu

Command and Control Engineering College

Army Engineering University of PLA, Nanjing, China

*Corresponding author: h_chen27@163.com, hzw_1983821@163.com

Abstract—With the rise of crowdsourced software testing in recent years, the issuers of crowd test tasks can usually collect a large number of test reports after the end of the task. These reports have insufficient validity and completeness, and manual review often takes a lot of time and effort. The crowdsourced test task publisher hopes that after the crowdsourced platform collects the test report, it can analyze the validity and completeness of the report to determine the severity of the report and improve the efficiency of crowdsourced software testing. In the past ten years, researchers have used various technologies (such as natural language processing, information retrieval, machine learning, deep learning) to assist in analyzing reports to improve the efficiency of report review. We have summarized the relevant literature of report analysis in the past ten years, and then classified from report classification, duplicate report detection, report prioritization, report refactoring, and summarized the most important research work in each area. Finally, we propose research trends in these areas and analyze the challenges and opportunities facing crowdsourced test report analysis.

Index Terms—Survey, report classification, duplicate report, report prioritization, report refactoring, crowdsourced testing

I. INTRODUCTION

In recent years, crowdsourced testing has become increasingly popular among software companies. After the software company develops the product, it will deliver the product to the crowdsourced platform, and the crowdsourced platform will release the product, recruit crowdsourced workers, and start the crowdsourced testing process. Crowdsourced workers fill out and submit test reports in accordance with the requirements of the crowdsourced testing platform to get some incentives (such as virtual currency, coupons). After the crowdsourced platform finishes the crowd testing process, it delivers a complete test report to the software company. The software company optimizes the product based on this final test report, thereby saving a lot of manpower and time.

Crowdsourced platforms usually have a special Test Report Management System(TRMS). These TRMSs automatically or semi-automatically analyze the test reports submitted by workers to reducing the pressure on the crowdsourced platform to review the test reports. Therefore, whether the TRMS can effectively analyze the test report submitted by the crowdsourced workers is very important for the crowdsourced platform to present a complete test report to the software company.

The purpose of this survey is two-fold: On the one hand, it is a more comprehensive survey on how to use test report analysis techniques and methods in crowdsourced testing. On

the other hand, it summarizes the effectiveness of existing test report analysis techniques in crowdsourced testing. Since crowdsourced testing is an emerging field in software testing, and various aspects of technology are still in the process of being perfected, our investigation seeks to find academic papers related to test report analysis. The techniques and methods proposed in these documents can be used directly or indirectly in the analysis of reports for crowdsourced testing, or a framework is designed to implement/improve the analysis of test reports.

The remaining chapters of this paper are as follows: Section 2 describes the literature sources and screening methods for this survey. Section 3 describes the general process of the test report analysis process of the crowdsourced testing platform, and the test report analysis technology and method research related to this process. Section 4 analyzes the challenges, trends and opportunities in crowdsourced test report analysis. Finally, we conclude in Section 5.

II. LITERATURE SOURCES AND SCREENING METHODS

The term “crowdsourcing” was coined by “crowd” and “outsourcing” by Jeff Howe and published in 2006[1]. The researchers then introduced the concept of crowdsourcing into software engineering and proposed crowdsourced software engineering[2]. And in crowdsourced software engineering, some researchers specialize in subdividing crowdsourced software testing areas[3]. Although the field of crowdsourced software testing has been relatively short, the concept and method of crowdsourced software testing has long been active in open source software communities and application stores. In order to explore the effectiveness of the test report analysis technology in the field of crowdsourced software testing in more detail, we collected the following processes for published papers:

First, we discussed the objects and scope of the thesis collection. In crowdsourced software testing descriptions, “test report” and “bug report” are often cross-used, and “bug report” or “defect report” is more common in the open source software community. In order to increase the breadth of the literature collection, we have also included the literature described as “bug report” and “defect report”, and we have also limited the time range from January 2010 to January 2020.

We then collected publications from this decade in three steps:

TABLE I
TERMS FOR FUZZY SEARCH

Category	Terms
Step 1	(crowd OR crowdsourcing OR crowdsourced) AND (test OR bug OR defect) AND report
Step 2	(test OR bug OR defect) AND report AND analysis
	(test OR bug OR defect) AND report AND classification
	(test OR bug OR defect) AND report AND duplicate
	(test OR bug OR defect) AND report AND prioritization
	(test OR bug OR defect) AND report AND summarization
	(test OR bug OR defect) AND report AND aggregation

- We perform fuzzy searches in five popular online academic literature search engines: *ACM Digital Library*, *IEEE Xplore Digital Library*, *Springer Link Online Library*, *Elsevier ScienceDirect* and *Google Scholar*. The keywords we used for fuzzy search are listed in Table I. We hope that the first batch of literature collected is closely related to the analysis of test reports in crowdsourced software testing.
- Due to the short time of crowdsourced software testing, we relaxed the search criteria and did not include words like crowdsourced software testing. However, the modifiers such as “classification”, “duplicate”, “prioritization”, and “summarization” are added. We hope that the second batch of collected documents can be used in the test report analysis of crowdsourced software testing.
- We performed manual citation screening of the literature collected in the first two steps, quickly browsing the title and abstract of the literature, and determining whether it is relevant to the analysis of the test report. We hope that the third batch of papers will be complementary.

After collating the collected literature, we finally got 102 articles related to the analysis of test reports(Fig. 1).In the process of selecting articles, we also found other studies of test reports, such as the use of tests for defect prediction, and the use of test reports for bug location. However, the subject of this survey is test reports, the purpose of which is to improve the efficiency of test report analysis in crowdsourced testing, so they will not be included in the survey.

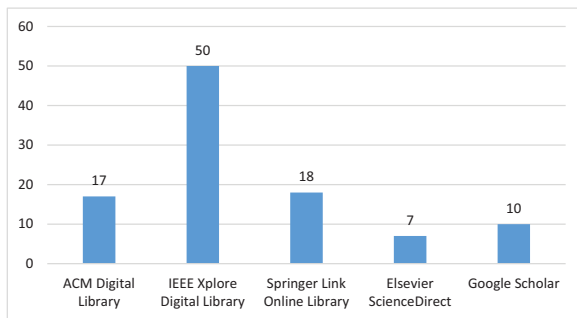




Fig. 1. Number of papers retrieved in each library.

TABLE II
AN EXAMPLE OF A REAL CROWDSOURCING TEST REPORT

Testing Case	Case-Id: 25567 Worker-Id: 15067 Title: <i>XXX Mobile Application Test Report Example</i>
Environment	Phone type: <i>R9 OPPO</i> Operation system: <i>Android 5.5</i>
Category	Type: <i>Incomplete function</i> level: <i>Emergency</i> frequency: <i>Always</i>
Test Case Description	1. Enter the integral mall according to the steps of function use 2. Check out the mall products 3. Select goods with less than 110 points and goods with more than 110 points for exchange 4. Check the rules for points 5. Check your exchange records 6. Check out more items
Result Description	The exchange record of the points mall cannot show the information of the goods that have been exchanged. It shows that the goods are in possession. There is no corresponding exchange record in the exchange record.
Screenshot	(a)  (b) 

III. TEST REPORT ANALYSIS PROCESS AND METHOD

Zhang et al.[3] have statistics on the existing common crowdsourced test platforms, and we summarize the processing steps after these crowdsourced platforms(such as Baidu crowdsourced test platform, Mootest platform, QQ crowdsourced test platform, etc.) collect test reports, and propose a general analysis process for crowdsourced test reports(Fig. 2).

The crowdsourced testing worker submits a test report on the crowdsourced platform, and the crowdsourced platform collects the test report of the crowdsourced testing task and stores it in the platform database, and we show a report example in Table II. After the task is completed, the test report submitted by the task is analyzed as follows:

a) *Report Classification*: Different test reports may reveal the same or different types of defects, so the test reports need to be classified. The use of automated classification technology can shorten the test report classification time, thereby improving the analysis efficiency of test reports.

b) *Duplicate Report Detection*: In the same type of test report, different test reports will report the same bugs, making the reports very similar. The crowdsourced platform hopes to be able to automatically identify these extremely similar reports and reduce the number of platform review reports, thereby improving the efficiency of test report analysis.

c) *Report Prioritization*: The test reports submitted by different crowdsourced workers, if they describe the same bug, the crowdsourced platform hopes to give priority to review the complete and clear test report; if they describe different bugs, the crowdsourced platform wants to be able to expose the software test reports of serious bugs are reviewed

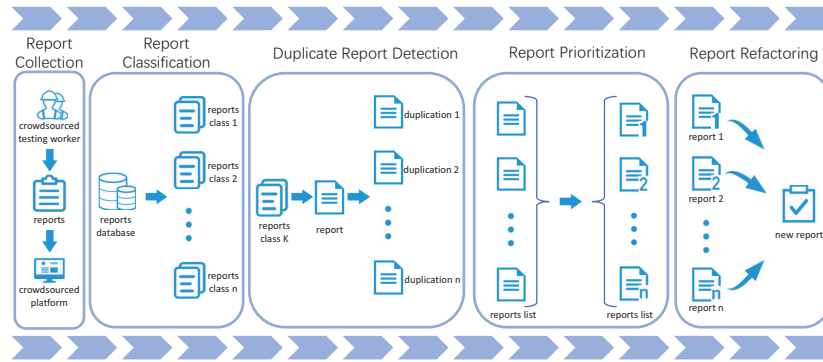


Fig. 2. Crowdsourced platform test report analysis general process.

first. Automating the prioritization of test reports can enable crowdsourcing platforms to increase the efficiency of test report review in a limited time.

d) *Report Refactoring*: After screening out some high-quality test reports, the crowdsourced platform hopes to supplement the high-quality test reports with redundant test reports, so that these natural language descriptions or test reports with pictures can highlight problems in the software use process, making platform workers can quickly view important information of test reports, improving the efficiency of test report analysis on crowdsourced platforms.

We divided the collected documents into domains(Fig. 3), and separated technical articles and review articles in their respective fields.

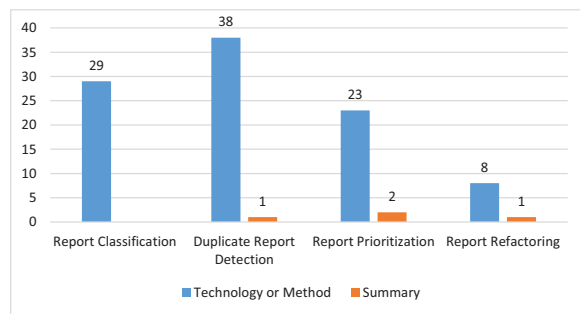


Fig. 3. Number of papers in each area.

A. Report Classification

Crowdsourced test report classification has two functions: one is to classify valid test reports, and the other is to cluster test reports that describe real bugs. In order to improve the effectiveness of the crowdsourced test report classification, in the report classification research, we focused on the datasets used by the researchers' experiments, the methods used in the experiments, and the evaluation metrics of the experimental results.

Datasets: We summarized the data set used in the report classification (Table. III) and the number of reports used in the

experiment (Fig.4).Most of the reports come from the open source software community, and these reports are managed by Bug-Tracking System, such as Eclipse, Mozilla. Some of the reports come from the crowdsourced test platform, which collects real crowdsourced test report data through publishing tasks. Some of the experimental data comes from comments from users in the App Store, like the Google Play Store and apple's App Store. About 83 percent of the researchers used less than or equal to 100,000 data, and more than half used less than or less than 10,000 data for the experiment, a number we believe is consistent with the crowdsourced test platform's collection of cross-project test reports and single-project test reports.

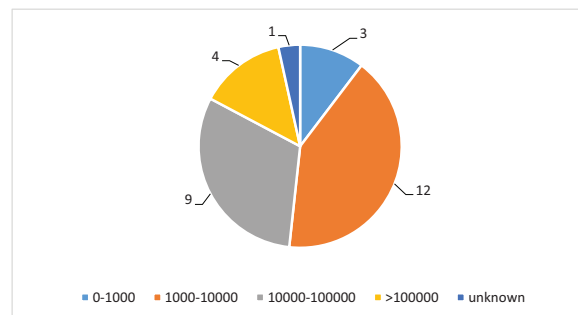


Fig. 4. Number of reports used in the experiment.

Methods: We collated the methods used by researchers in the classification of the report(Table IV). We are broadly divided into six broad categories: N-gram Model, Machine Learning, Information, Topic Model, Graph Model, and Data Reduction. N-gram Model is used to generate text features and text matching, Machine Learning methods are used to train classifiers, Information Retrieval and Topic Models are used to extract theme features of documents or document classes, Graph Models are used to process bug reports with stack information, and Data Reduction technology is used to select high-quality reports and features. We found that the researchers focused their report classification study on two types of issues, one classifying reports as Bug or Un-

TABLE III
NUMBER OF PAPERS USING EACH DATASET

Dataset	Papers	Dataset	Papers
Eclipse	12	Netbeans	2
Mozilla	7	OpenFOAM	2
Crowdsourced testing platform	6	ArgoUML	1
Google Play store	4	Bugzilla	1
Firefox	3	F-Droid	1
Jboss	3	GNOME	1
Lucene	3	Mahout	1
Apples App Store	2	Mylyn	1
GCC	2	OpenNLP	1
HTTPClient	2	OpenOffice	1
Jackrabbit	2	Twitter	1
Microsoft Project	2		

Bug, and the other classifying reports and recommending them to appropriate developers. Pandey et al.[4] compared the impact of six different classification algorithms on the performance of bug report classification(Bug or Un-Bug). Tian et al.[5] combines the text information of the bug report and developer activity characteristics to train a classifier in order to recommend the new report to the developer with the highest probability of solving. Guzman et al.[6] and Panichella et al.[7] studies showed that integrating multiple machine learning classifiers outperformed a single machine learning classifier.

Evaluation Metrics: We reviewed the evaluation metrics used by researchers to evaluate the performance of classification methods. The results show that most researchers use Precision, Recall, and F-measure to evaluate classification methods. Precision indicates the number of positive class predictions that actually belong to the positive class, Recall indicates the number of positive class predictions made out of all positive examples in the dataset, F-measure is the weighted harmonic average of Precision and Recall. They are defined as follows:

$$Precision = \frac{TP}{TP + FP} \quad (1)$$

$$Recall = \frac{TP}{TP + FN} \quad (2)$$

$$F - measure = (\beta^2 + 1) * \frac{Precision * Recall}{\beta^2 * Precision + Recall} \quad (3)$$

where TP denote the number of reports correctly labeled to a class, FP denote the number of reports incorrectly labeled to a class, FN denote the number of reports that belong to a class are divided into another class, β is a hyperparameter used to adjust the weights of *Precision* and *Recall*. We record *F-measure* as *F1* when β is 1. Accuracy is sometimes used to evaluate classifier performance, and it is defined below:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (4)$$

TABLE IV
METHODS USED BY RESEARCHERS

Technology	Method	References
N-gram Model	Character-level	[8]
	Word-level	[9],[10]
Machine Learning	Support Vector Machine	[11]
	Naive Bayes	[9],[12],[10]
	Expectation-Maximization	[13],[12]
	K-Means	[14],[15]
	X-Means	[13]
	Active Learning	[14],[16]
	Self Training	[14]
	Multinomial Naive Bayes	[17],[18]
	Gradient Boosted Regression Trees	[19]
	K-Modes	[20]
	Stacked Denoising Autoencoder	[21]
	Fuzzy Clustering	[22]
	Choquet Fuzzy Integral	[23]
Convolutional Neural Networks	[24]	
Spatial Pyramid Matching	[25]	
Information Retrieval	-	[19]
Topic Model	Latent Dirichlet Allocation	[13],[26]
	Hierarchical Dirichlet process	[13]
	Bitern Topic Model	[18]
Graph Model	Crash Graph Matching	[27]
	Call Stack Matching	[28]
	Social Network	[29]
	Developer-Component-Bug Network	[30]
	Bayesian Network	[17]
Data Reduction	Feature Selection	[31],[9],[32]
	Instance Selection	[31],[32]

where TN denote the number of reports that do not belong to a class and are not classified in this class. *Accuracy* is the most intuitive performance measure, but classifiers cannot be well evaluated using *Accuracy* alone. Especially when some datasets are unevenly distributed, we need to consider multiple indicators to evaluate the classifier.

B. Duplicate Report Detection

Crowdsourced duplicate test report detection is to avoid double review of test reports that describe the same bug. Swapna et al.[33] did an investigation on duplicate bug detection. The duplicate bug detection determines whether it is the same bug by analyzing the information contained in the bug report. We follow its ideas, summarize the contributions made by researchers in the process of detecting duplicate reports(Table V), and collate the report content, analysis methods, similarity measurements, and evaluation metrics used in the experiments.

Report Content: We counted the content of reports used by researchers in the experiment(Table VI). Report formats for different data sets are usually quite different. We found that for test reports containing summary and description fields, almost all researchers have used these field information. Contextual information will also be used to assist detection. Sun et al.[37] earlier used product, component, priority and other information to construct a retrieval function REP for duplicate report detection. Ebrahimi et al.[50][60] focus on using stack trace to detect reports describing the same bug.

TABLE V
LIST OF CONTRIBUTIONS BY RESEARCHERS

Type	References
Tools	[34],[35]
Methods	[36],[37],[38],[39],[40],[41],[42],[43],[44],[45],[46],[47],[48],[49],[50],[51],[52],[53],[54],[55],[56],[57],[58],[59],[60],[61],[62],[63]
Comparisons	[64],[65],[66]
Frames	[67]
Features	[68],[69],[70]
Datasets	[71]

TABLE VI
NUMBER OF CONTENT USED BY RESEARCHERS

Information Type	Content	Number of papers
Textual Information	Summary	20
	Description	32
	Title	10
	Comment	2
Contextual Information	Product	3
	Component	6
	Version	4
	E-mail	1
	Tag	3
	Type	3
	Priority	3
	Status	2
Date	2	
Log Information	Stack Trace	4
Image Information	Screenshot	2

Analysis Methods: During the experiment, scholars tried different methods to deal with different types of field information. In order to transform natural language information into data capable of calculating similarity, TF-IDF is the most used method. Given a term t and a document d in a dataset D , it is defined as follows:

$$TF(t, d) = \frac{\text{Number of terms } t \text{ appears in } d}{\text{Number of terms in } d} \quad (5)$$

$$IDF(t) = \log \frac{\text{Number of documents in } D}{\text{Number of documents containing } t} \quad (6)$$

$$TF - IDF(t, d) = TF(t, d) \times IDF(t) \quad (7)$$

The main idea of TF-IDF is that if a term appears frequently in one document, but rarely in all the documents, then the term has a high TF-IDF value and is suitable for classification. Some researches also use word embedding to transform text information[54][55][58][59][63][66]. This is the most commonly used method in natural language processing at present, it preserves some of the connections between words when converting text to vectors. In order to convert the screenshot into a vector representation, Wang et al.[58][63] used Gist descriptor and MPEG-7 descriptor to extract the structure feature and color feature of the image respectively for the screenshot information.

Similarity Measurements: In document vector space, most researchers use cosine similarity to measure document similarity. Cosine similarity is used to calculate the cosine of the angle between two high-dimensional vectors in the vector space. Given two document vectors \vec{a} and \vec{b} , the calculation formula is as follows:

$$\text{Cosine similarity} = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \times \|\vec{b}\|} \quad (8)$$

Some scholars have tried to use Manhattan distance in experiments to measure the similarity of topic features extracted from documents[62]. It is defined as:

$$\text{Manhattan distance}(X, Y) = \sum_{i=1}^n |x_i - y_i| \quad (9)$$

where X and Y are two topic feature vector. Feng et al.[44] also uses Jensen-Shannon and symmetric KL divergences to calculate the similarity of document topics. The BM25 and BM25F algorithms based on probabilistic retrieval models are also commonly used for document similarity retrieval. The BM25 search score between a query string q and a document d is calculated as follows:

$$s(q, d) = \sum_{t \in q} idf(t) \cdot \frac{occurs_t^d}{k_1((1-b) + b \frac{l_d}{avl_d}) + occurs_t^d} \quad (10)$$

where $occurs_t^d$ is the term frequency of t in d ; l_d is the document d length; avl_d is the average length of the document in the dataset; k_1 is a free parameter and $b \in [0, 1]$. The $idf(t)$ denotes the inverse document frequency of the query term t , it is computed by the following equation:

$$idf(t) = \log \frac{N - df(t) + 0.5}{df(t) + 0.5} \quad (11)$$

Where N is the total number of documents in dataset; $df(t)$ is the number of documents containing the term t . BM25F has made some improvements in BM25, which is not only considering words as individuals, but also dividing documents into individuals according to fields. Sun et al.[37] extend BM25F into the following form by considering the term frequencies W_Q in queries:

$$BM25F_{ext}(q, d) = \sum_{t \in (d \cap q)} IDF(t) \times \frac{TF_D(d, t)}{k_1 + TF_D(d, t)} \times W_Q$$

$$\text{where } TF_D(d, t) = \sum_{f=1}^K \frac{w_f \times occurs_t^{d[f]}}{1 - b_f + \frac{b_f \times l_f}{avl_f}},$$

$$W_Q = \frac{(k_2 + 1) \times TF_Q(q, t)}{k_2 + TF_Q(q, t)},$$

$$TF_Q(q, t) = \sum_{f=1}^K w_f \times occurs_t^{q[f]} \quad (12)$$

In (12), given a dataset of N documents, each document d consists of K fields, and $d[f]$ denote the terms bag in the f -th field. For each field f , w_f is its field weight; $occurs_t^{d[f]}$

is the number of occurrences of term t in field f ; l_f is the size of the bag $d[f]$; avl_f is the average size of the bag $d[f]$ across all documents in dataset; b_f and k_2 are free parameter and $b_f, k_2 \in [0, 1]$. For the query, $q[f]$ denote the terms bag in the f -th field and $occurs_t^{q[f]}$ is the number of occurrences of term t in field f . In recent years, researchers have built complex machine learning models, such as HMM[50][60], CNN[59][66], RNN[66] and LSTM[66], hoping that the model can accept document vectors and mine potential features to calculate and output similarity.

Evaluation Metrics: Duplicate report detection returns the K scores with the highest score by calculating the similarity between the input report and the historical report. Most researchers use Recall@K and Mean Average Precision (MAP) to evaluate the performance of the duplicate report detection algorithm. The Recall@k calculates the recall rate of the recommended list of copies of length k, it is defined as follows:

$$Recall@K = \frac{hit@k}{hit@k + missed@k} \quad (13)$$

where $hit@k$ represents the number of duplicate reports in the recommendation list of length k, and $missed@k$ is the number of duplicate reports that did not appear on the candidate list. The MAP is to average the AP corresponding to multiple queries, for each query, the AP is calculated as follows:

$$AveP = \frac{\sum_{idx=1}^k P(idx) \times rel(idx)}{Number\ of\ duplicate\ reports} \quad (14)$$

where idx is the sorting position in the retrieval result queue, and $P(idx)$ is the accuracy of the first idx results; $rel(idx)$ is 1 if idx -th is the duplicate report else 0. And for multiple N queries, the MAP is calculated as follows:

$$MAP = \frac{\sum_{q=1}^N AveP(q)}{N} \quad (15)$$

Some researchers also use Mean Reciprocal Rank (MRR) to assess the quality of the list of duplicate reports returned[42][54][58][61][63][66]. The core idea of MRR is to return a result set with the first correct answer near the top and the better the result set. For the N queries, it is defined as:

$$MRR = \frac{1}{N} \sum_{q=1}^N \frac{1}{rank_q} \quad (16)$$

where $rank_q$ is the index of first correct answer in the q -th query. Banerjee et al.[69] used the pROC package to draw the Receiver Operating Characteristic (ROC) curve and used Area Under Curve (AUC) to evaluate their proposed method using 24 features to calculate similarity.

C. Report Prioritization

Crowdsourced test report prioritization from a bug fix perspective, the hope of early detection of high-risk bugs, timely repair to reduce losses, from the report review perspective, the hope to be able to review the content of the complete and well-described test reports, so as to accurately identify the defects

TABLE VII
FEATURE EXTRACTION METHODS USED BY RESEARCHERS

Methods	References
TF-IDF & Information Gain	[77],[78],[79]
Basic Features & Basic and Predicted Features	[80]
BM25Fext & REP	[81]
TF & contextual features	[82]
22 new features	[83],[84]
Keyword Vector & Risk Vector	[85]
Keyword Vector & Image Feature Histogram	[86]
Time Features & Basic Features	[74]
Importance Degree Reduction & Keyword Vector	[87]
Word Embedding	[88]
Instance Selection & Feature Selection	[89],[90]
Entropy & Summary Weight	[91]
TF & Emotion-Value	[92]
Control-flow Automaton & Access Path & Critical Functions	[93]
TF-IDF	[94]
TF & Information Gain & Chi-square	[95]
Word Embedding & Emotion-Value	[75],[76]
TF & Information Gain & Topic Proportions Vector	[96]

described in the report. Gomes et al.[72] and Uddin et al.[73] started study on the security analysis and prioritization of bug reports. We followed their research methods and combed the feature extraction methods, experimental methods and evaluation methods used by researchers in recent years.

Feature Extraction: We list the feature extraction methods used by scholars(Table VII). Almost all researchers have extracted the information features of this article from the report description or summary, and some studies have made feature selection in the extracted features to reduce the feature space. Xu et al.[74] extra extracted the time characteristics of the test report for report prioritization experiments. Ramay et al.[75] and Umer et al.[76] analysis reports describe the sentiment scores to aid priority determination.

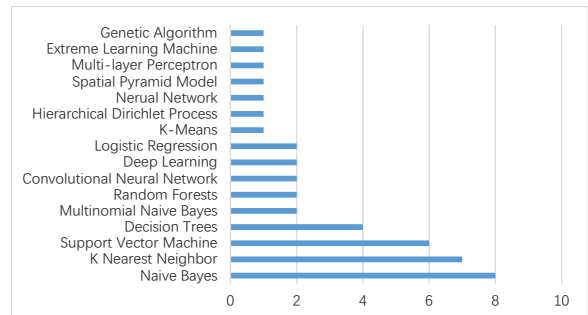


Fig. 5. List of methods used in the experiment.

Experimental Methods: We counted the use of experimental methods by researchers(Fig. 5). Most scholars regard report

prioritization as a special classification problem and classify test reports into prioritized priority categories. But the distance between classes is not equal, and the gap between two adjacent priority classes is always smaller than the distance between non-adjacent classes. Feng et al.[85] proposes a priority ranking method called DivRisk, it uses a diversity strategy and a risk strategy to evaluate the test report, calculates a priority value for the test report, and sorts according to the priority value. Feng et al.[86] also uses a diversity strategy in prioritizing test reports with screenshots. Ramay et al.[75] and Umer et al.[76] apply deep learning techniques combined with sentiment scores to report prioritization.

Evaluation Metrics: When researchers consider priority ranking as a classification problem, Precision, Recall, and F-measure are used to evaluate the effectiveness of the method. Feng et al.[85][86] evaluation proposes a priority ordering algorithm using the Average Percentage of Fault Detected (APFD), which is defined as follows:

$$APFD = 1 + \frac{T_{f1} + T_{f2} + \dots + T_{fM}}{n \times M} + \frac{1}{2 \times n} \quad (17)$$

in which, n is the number of test reports and M is the total number of faults reported in all test reports. T_{fi} is the index of the first test report that revealed failure i . Some researchers draw the Receiver Operating Characteristic (ROC) curve and calculate the Area Under Roc Curve (AUC) evaluation model[87][79][94][90]. The ROC space defines the False Positive Rate (FPR) as the X axis and the True Positive Rate (TPR) as the Y axis. FPR and TPR are calculated as follows:

$$FPR = \frac{FP}{FP + TN} \quad (18)$$

$$TPF = \frac{TP}{TP + FN} \quad (19)$$

Then calculate the area from the curve.

D. Report Refactoring

The crowdsourced test report refactoring is to help developers quickly understand the defects described in the report. Tarar et al.[97] made a brief review of the bug report summary, and we made a more detailed division based on it. We divided the report reconstruction tasks into summarization, generation, and enhancement, and summarized their respective experimental methods, and finally explained their evaluation metrics.

Summarization: The report summarization expresses the content of the report by selecting a summary sentence. Rastkar et al.[98] uses the existing dialog-based auto-summarizer to train on the bug report corpus, by scoring the input report sentences, and then selecting sentences to form a summary according to the score from high to low, until the number of words reaches the compression ratio. Jha et al.[99] proposed the Bag-of-Frames data representation method, which is different from the common Bag-of-Words representation method, and cross-contrasts using the five summarization algorithms (Random, Hybrid TF, Hybrid TF-IDF, SumBasic,

and LexRank). The experimental results show that using BOF representation for classification and BOW representation for the summary can get the most accurate results, and SumBasic can generate a summary that is largely consistent with human judgment. Li et al.[100] built a deep learning model DeepSum to summarize the report. The model received a new bug report and retrieved a set of reports similar to the new report in the historical data, then use this set of defect reports to train a stepped autoencoder network to score the newly reported sentences, and finally use dynamic programming to extract summary sentences.

Generation: For bug reports containing picture information, researchers hope to be able to assist in understanding reports for converting pictures into descriptive text information. Liu et al.[101] uses SPM to measure image similarity, then selects high-quality reports containing similar pictures, and uses natural language processing technology to build language models to analyze text descriptions of high-quality reports, generate descriptive keywords for pictures, and help developers understand the test report. Yu et al.[102] builds a deep learning model CroReG to generate a text report from the picture. The model uses OCR technology to identify the text information in the picture. At the same time, it uses a CNN network as an encoder to extract features from the picture, and then uses an LSTM network as a decoder to convert the picture features. The vector is translated into a natural language defect description, and finally a test report is generated by combining the OCR-recognized text information and decoded information.

Enhancement: Test report enhancements supplement the main test report by extracting useful information from the duplicate report, thereby reducing the number of report reviews and improving the quality of report reviews. Chen et al.[103] proposed a new test report extension framework, which uses merge operation, TF-IDF-based algorithm and graph model to enhance the environment field, input field and description field of test report respectively, and highlights the supplementary information through visualization technology. Hao et al.[104] and Li et al.[105] designed and implemented the report enhancement tool CTRAS. CTRAS used PageRank algorithm to obtain the master report, weighted the text description and picture information of similar reports, and then extracted the information fragments contained in the high score report but not in the master report for supplement.

Evaluation Metrics: Precision, Recall, F-measure and Pyramid scores are widely used to evaluate sentence and keyword choices in report reconstruction. Given a set of summary sentences s_{select} , the Pyramid score is calculated as follows:

$$Pyramid = \frac{Num_{TotalLinks}}{Num_{MaxLinks}} \quad (20)$$

where $Num_{TotalLinks}$ denote the number of times that sentences linked by the annotators in s_{select} and $Num_{MaxLinks}$ is the maximum possible links for the same number of sentences. And some researchers realistically evaluate model performance by comparing the test reports generated by the model with the artificially aggregated test reports[99][103].

IV. CHALLENGES AND TRENDS AND OPPORTUNITIES

At present, the crowdsourced platform mainly faces three problems: First, a large number of test reports submitted by workers. The simplicity of crowdsourced software testing makes people actively participate in and submit reports to get paid. Large crowdsourced platforms can collect hundreds of thousands of reports on an average day. These reports originate from different projects, so the content of the collected reports is very different. For example, some of them contain pictures or log information, extracting effective information from a large set of reports is a challenge. Secondly, users are likely to encounter the same problems in the testing process, but the reports describing the same problems will also differ from each other. How to use the similarities and differences in reports to improve the test coverage in defect detection is also a challenge. Finally, the uneven knowledge and testing techniques of the workers' fields led them to produce different quality test reports when writing the reports, which affected the completeness and readability of the reports and severely hindered the review of the reports.

Many scholars now conduct research in the field of defect report analysis, remove invalid reports through classification technology, calculate similarity to determine similar reports, prioritize test reports, and aggregate test reports. We found that the methods used by researchers have evolved from earlier text matching to attempts to use machine learning to deep learning. The focus has also changed from the initial character features to keyword features to semantic features. Types of information are also tried to be processed using a variety of methods, such as image analysis, abstract syntax trees, graph models, etc. Data reduction techniques are also used to select high-quality features. From the experimental results, almost all studies show that weighted combination of multiple algorithms can achieve better results than a single algorithm. In the experiments using semantic features, they can often produce results that are more consistent with people's cognition.

Pre-trained language models have achieved amazing results in tasks related to natural language processing, such as: semantic similarity, machine translation, and reasoning questions and answers. For researchers, how to apply language models in each step of test report analysis to improve the efficiency of crowdsourcing test report analysis is of great value. For the crowdsourcing platform, it is meaningful to build a test report analysis process framework and combine different report analysis methods, and encapsulate the test report processing steps into a perfect tool to improve the efficiency of crowdsourcing software testing.

V. CONCLUSION

Crowdsourced software testing greatly reduces the effort and expense that software companies spend on software testing. But the pressure didn't go away, but moved to the crowdsourced test platform. The platform needs to process a large number of test reports every day. Processing these reports manually is too time-consuming and may affect the repair of important defects. To address this issue, it is necessary to

apply some automated or semi-automated analytical methods to assist in report review.

This paper summarizes a series of work in the field of test report analysis. In this survey, we first introduced some preparations. Secondly, we divided the bug report analysis and discussed the concerns in all directions. Third, we analyzed the challenges and feasible solutions faced by the crowdsourced test platform. Finally, we summarize the whole paper and put forward the future work direction.

This research mainly focuses on the effective methods proposed in the field of report analysis, and we plan to use pre-trained language models to improve test report classification and reconstruction in the future, hoping to help crowdsourced platform to improve test report analysis efficiency.

ACKNOWLEDGMENT

The project is supported by National Key R&D Program of China (No: 2018YFB1403400), Natural Science Foundation of China (No: 61702544), Natural Science Foundation of Jiangsu Province, China (No: BK20160769, BK20141072), China Postdoctoral Science Foundation (No: 2016M603031).

REFERENCES

- [1] Jeff Howe. The rise of crowdsourcing. *Wired magazine*, 14(6):1–4, 2006.
- [2] Ke Mao, Licia Capra, Mark Harman, and Yue Jia. A survey of the use of crowdsourcing in software engineering. *Journal of Systems and Software*, 126:57–84, 2017.
- [3] Xiaofang Zhang, Yang Feng, D Liu, Z Chen, and B Xu. Research progress of crowdsourced software testing. *Journal of Software*, 29(1):69–88, 2018.
- [4] Nitish Pandey, Debarshi Kumar Sanyal, Abir Hudait, and Amitava Sen. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering*, 13(4):279–297, 2017.
- [5] Yuan Tian, Dinusha Wijedasa, David Lo, and Claire Le Goues. Learning to rank for bug report assignee recommendation. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10. IEEE, 2016.
- [6] Emitza Guzman, Muhammad El-Haliby, and Bernd Bruegge. Ensemble methods for app review classification: An approach for software evolution (n). In *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 771–776. IEEE, 2015.
- [7] Sebastiano Panichella, Andrea Di Sorbo, Emitza Guzman, Corrado A Visaggio, Gerardo Canfora, and Harald C Gall. Ardoc: App reviews development oriented classifier. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 1023–1027, 2016.
- [8] Ashish Sureka and Pankaj Jalote. Detecting duplicate bug report using character n-gram-based features. In

- 2010 *Asia Pacific Software Engineering Conference*, pages 366–374. IEEE, 2010.
- [9] Nivir Kanti Singha Roy and Bruno Rossi. Towards an improvement of bug severity classification. In *2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 269–276. IEEE, 2014.
- [10] Ashima Kukkar and Rajni Mohana. A supervised bug report classification with incorporate and textual field knowledge. *Procedia computer science*, 132:352–361, 2018.
- [11] John Anvik and Gail C Murphy. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 20(3):1–35, 2011.
- [12] Jifeng Xuan, He Jiang, Zhilei Ren, Jun Yan, and Zhongxuan Luo. Automatic bug triage using semi-supervised text classification. *arXiv preprint arXiv:1704.04769*, 2017.
- [13] Nachai Limsettho, Hideaki Hata, Akito Monden, and Kenichi Matsumoto. Unsupervised bug report categorization using clustering and labeling algorithm. *International Journal of Software Engineering and Knowledge Engineering*, 26(07):1027–1053, 2016.
- [14] Ferdian Thung, Xuan-Bach D Le, and David Lo. Active semi-supervised defect categorization. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 60–70. IEEE, 2015.
- [15] Junjie Wang, Qiang Cui, Qing Wang, and Song Wang. Towards effectively test report classification to assist crowdsourced testing. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2016.
- [16] Junjie Wang, Song Wang, Qiang Cui, and Qing Wang. Local-based active classification of test report to assist crowdsourced testing. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, pages 190–201, 2016.
- [17] Yu Zhou, Yanxiang Tong, Ruihang Gu, and Harald Gall. Combining text mining and data mining for bug report classification. *Journal of Software: Evolution and Process*, 28(3):150–176, 2016.
- [18] Emitza Guzman, Mohamed Ibrahim, and Martin Glinz. A little bird told me: Mining tweets for requirements and software evolution. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*, pages 11–20. IEEE, 2017.
- [19] Adelina Ciurumelea, Andreas Schaufelbühl, Sebastiano Panichella, and Harald C Gall. Analyzing reviews and code of mobile apps for better release planning. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 91–102. IEEE, 2017.
- [20] Siyuan Shen, Hao Lian, Tieke He, and Zhenyu Chen. Clustering on the stream of crowdsourced testing. In *2017 14th Web Information Systems and Applications Conference (WISA)*, pages 317–322. IEEE, 2017.
- [21] Junjie Wang, Qiang Cui, Song Wang, and Qing Wang. Domain adaptation for test report classification in crowdsourced testing. In *2017 IEEE/ACM 39th International Conference on Software Engineering: Software Engineering in Practice Track (ICSE-SEIP)*, pages 83–92. IEEE, 2017.
- [22] He Jiang, Xin Chen, Tieke He, Zhenyu Chen, and Xiaochen Li. Fuzzy clustering of crowdsourced test reports for apps. *ACM Transactions on Internet Technology (TOIT)*, 18(2):1–28, 2018.
- [23] Rong Chen, Shi-Kai Guo, Xi-Zhao Wang, and Tian-Lun Zhang. Fusion of multi-rsmote with fuzzy integral to classify bug reports with an imbalanced distribution. *IEEE Transactions on Fuzzy Systems*, 27(12):2406–2420, 2019.
- [24] I Made Mika Parwita and Daniel Siahaan. Classification of mobile application reviews using word embedding and convolutional neural network. *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi*, pages 1–8, 2019.
- [25] Yan Yang, Xiangjuan Yao, and Dunwei Gong. Clustering study of crowdsourced test report with multi-source heterogeneous information. In *International Conference on Data Mining and Big Data*, pages 135–145. Springer, 2019.
- [26] Xin Xia, David Lo, Ying Ding, Jafar M Al-Kofahi, Tien N Nguyen, and Xinyu Wang. Improving automated bug triaging with specialized topic model. *IEEE Transactions on Software Engineering*, 43(3):272–297, 2016.
- [27] Sunghun Kim, Thomas Zimmermann, and Nachiappan Nagappan. Crash graphs: An aggregated view of multiple crashes to improve crash triage. In *2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN)*, pages 486–493. IEEE, 2011.
- [28] Yingnong Dang, Rongxin Wu, Hongyu Zhang, Dongmei Zhang, and Peter Nobel. Rebucket: A method for clustering duplicate crash reports based on call stack similarity. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 1084–1093. IEEE, 2012.
- [29] Tao Zhang and Byungjeong Lee. An automated bug triage approach: A concept profile and social network based developer recommendation. In *International Conference on Intelligent Computing*, pages 505–512. Springer, 2012.
- [30] Hao Hu, Hongyu Zhang, Jifeng Xuan, and Weigang Sun. Effective bug triage based on historical bug-fix information. In *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pages 122–132. IEEE, 2014.
- [31] Weiqin Zou, Yan Hu, Jifeng Xuan, and He Jiang. Towards training set reduction for bug triage. In *2011 IEEE 35th Annual Computer Software and Applications Conference*, pages 576–581. IEEE, 2011.
- [32] Jifeng Xuan, He Jiang, Yan Hu, Zhilei Ren, Weiqin Zou,

- Zhongxuan Luo, and Xindong Wu. Towards effective bug triage with software data reduction techniques. *IEEE transactions on knowledge and data engineering*, 27(1):264–280, 2014.
- [33] D Swapna and K Thammi Reddy. A study of information retrieval approaches in duplicate bug detection. *Indian J. Sci. Technol*, 9(43), 2016.
- [34] Yoonki Song, Xiaoyin Wang, Tao Xie, Lu Zhang, and Hong Mei. Jdf: detecting duplicate bug reports in jazz. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 315–316. IEEE, 2010.
- [35] Ferdian Thung, Pavneet Singh Kochhar, and David Lo. Dupfinder: integrated tool support for duplicate bug report detection. In *Proceedings of the 29th ACM/IEEE international conference on Automated software engineering*, pages 871–874, 2014.
- [36] Tomi Prifti, Sean Banerjee, and Bojan Cukic. Detecting bug duplicate reports through local references. In *Proceedings of the 7th International Conference on Predictive Models in Software Engineering*, pages 1–9, 2011.
- [37] Chengnian Sun, David Lo, Siau-Cheng Khoo, and Jing Jiang. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, pages 253–262. IEEE, 2011.
- [38] Sean Banerjee, Bojan Cukic, and Donald Adjero. Automated duplicate bug report classification using subsequence matching. In *2012 IEEE 14th International Symposium on High-Assurance Systems Engineering*, pages 74–81. IEEE, 2012.
- [39] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N Nguyen, David Lo, and Chengnian Sun. Duplicate bug report detection with a combination of information retrieval and topic modeling. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*, pages 70–79. IEEE, 2012.
- [40] Yuan Tian, Chengnian Sun, and David Lo. Improved duplicate bug report identification. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 385–390. IEEE, 2012.
- [41] Cheng-Zen Yang, Hung-Hsueh Du, Sin-Sian Wu, and Xiang Chen. Duplication detection for software bug reports based on bm25 term weighting. In *2012 Conference on Technologies and Applications of Artificial Intelligence*, pages 33–38. IEEE, 2012.
- [42] Jian Zhou and Hongyu Zhang. Learning to rank duplicate bug reports. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 852–861, 2012.
- [43] Sean Banerjee, Zahid Syed, Jordan Helmick, and Bojan Cukic. A fusion approach for classifying duplicate problem reports. In *2013 IEEE 24th International Symposium on Software Reliability Engineering (ISSRE)*, pages 208–217. IEEE, 2013.
- [44] Liang Feng, Leyi Song, Chaofeng Sha, and Xueqing Gong. Practical duplicate bug reports detection in a large web-based development community. In *Asia-Pacific Web Conference*, pages 709–720. Springer, 2013.
- [45] Johannes Lerch and Mira Mezini. Finding duplicates of your yet unwritten bug report. In *2013 17th European Conference on Software Maintenance and Reengineering*, pages 69–78. IEEE, 2013.
- [46] Raj P Gopalan and Aneesh Krishna. Duplicate bug report detection using clustering. In *2014 23rd Australian Software Engineering Conference*, pages 104–109. IEEE, 2014.
- [47] Chao-Yuan Lee, Dan-Dan Hu, Zhong-Yi Feng, and Cheng-Zen Yang. Mining temporal information to improve duplication detection on bug reports. In *2015 IIAI 4th International Congress on Advanced Applied Informatics*, pages 551–555. IEEE, 2015.
- [48] Yun Zhang, David Lo, Xin Xia, and Jian-Ling Sun. Multi-factor duplicate question detection in stack overflow. *Journal of Computer Science and Technology*, 30(5):981–997, 2015.
- [49] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. Mining duplicate questions of stack overflow. In *2016 IEEE/ACM 13th Working Conference on Mining Software Repositories (MSR)*, pages 402–412. IEEE, 2016.
- [50] K Neda Ebrahimi, Md Shariful Islam, Abdelwahab Hamou-Lhadj, and Mohammad Hamdaqa. An effective method for detecting duplicate crash reports using crash traces and hidden markov models. In *Proceedings of the 26th Annual International Conference on Computer Science and Software Engineering*, pages 75–84. IBM Corp., 2016.
- [51] Abram Hindle, Anahita Alipour, and Eleni Stroulia. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering*, 21(2):368–410, 2016.
- [52] Kasthuri Jayarajah, Meera Radhakrishnan, and Camellia Zakaria. Duplicate issue detection for the android open source project. In *Proceedings of the 5th International Workshop on Software Mining*, pages 24–31, 2016.
- [53] Meng-Jie Lin, Cheng-Zen Yang, Chao-Yuan Lee, and Chun-Chang Chen. Enhancements for duplication detection in bug reports with manifold correlation features. *Journal of Systems and Software*, 121:223–233, 2016.
- [54] Xinli Yang, David Lo, Xin Xia, Lingfeng Bao, and Jianling Sun. Combining word embedding with information retrieval to recommend similar bug reports. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pages 127–137. IEEE, 2016.
- [55] Jayati Deshmukh, Sanjay Podder, Shubhashis Sengupta, Neville Dubash, et al. Towards accurate duplicate bug retrieval using deep learning techniques. In *2017 IEEE International conference on software maintenance and*

- evolution (ICSME)*, pages 115–124. IEEE, 2017.
- [56] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E Hassan. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *IEEE Transactions on Software Engineering*, 44(12):1245–1268, 2017.
- [57] Amar Budhiraja, Raghu Reddy, and Manish Shrivastava. Lwe: Lda refined word embeddings for duplicate bug report detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceedings*, pages 165–166, 2018.
- [58] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies, and Qing Wang. Cutting away the confusion from crowdtesting. *arXiv preprint arXiv:1805.02763*, 2018.
- [59] Qi Xie, Zhiyuan Wen, Jieming Zhu, Cuiyun Gao, and Zibin Zheng. Detecting duplicate bug reports with convolutional neural networks. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 416–425. IEEE, 2018.
- [60] Neda Ebrahimi, Abdelaziz Trabelsi, Md Shariful Islam, Abdelwahab Hamou-Lhadj, and Kobra Khanmohammadi. An hmm-based approach for automatic detection and classification of duplicate bug reports. *Information and Software Technology*, 113:98–109, 2019.
- [61] Abram Hindle and Curtis Onuczko. Preventing duplicate bug reports by continuously querying bug reports. *Empirical Software Engineering*, 24(2):902–936, 2019.
- [62] Behzad Soleimani Neysiani and Seyed Morteza Babamir. New methodology for contextual features usage in duplicate bug reports detection: Dimension expansion based on manhattan distance similarity of topics. In *2019 5th International Conference on Web Research (ICWR)*, pages 178–183. IEEE, 2019.
- [63] Junjie Wang, Mingyang Li, Song Wang, Tim Menzies, and Qing Wang. Images don’t lie: Duplicate crowdtesting reports detection with screenshot information. *Information and Software Technology*, 110:139–155, 2019.
- [64] Nilam Kaushik and Ladan Tahvildari. A comparative study of the performance of ir models on duplicate bug detection. In *2012 16th European Conference on Software Maintenance and Reengineering*, pages 159–168. IEEE, 2012.
- [65] Annibale Panichella. A systematic comparison of search algorithms for topic modelling—a study on duplicate bug report identification. In *International Symposium on Search Based Software Engineering*, pages 11–26. Springer, 2019.
- [66] Liting Wang, Li Zhang, and Jing Jiang. Detecting duplicate questions in stack overflow via deep learning approaches. In *2019 26th Asia-Pacific Software Engineering Conference (APSEC)*, pages 506–513. IEEE, 2019.
- [67] Mehdi Amoui, Nilam Kaushik, Abraham Al-Dabbagh, Ladan Tahvildari, Shimin Li, and Weining Liu. Search-based duplicate defect detection: an industrial experience. In *2013 10th Working Conference on Mining Software Repositories (MSR)*, pages 173–182. IEEE, 2013.
- [68] Nathan Klein, Christopher S Corley, and Nicholas A Kraft. New features for duplicate bug detection. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 324–327, 2014.
- [69] Sean Banerjee, Zahid Syed, Jordan Helmick, Mark Culp, Kenneth Ryan, and Bojan Cucic. Automated triaging of very large bug repositories. *Information and software technology*, 89:1–13, 2017.
- [70] Oscar Chaparro, Juan Manuel Florez, Unnati Singh, and Andrian Marcus. Reformulating queries for duplicate bug report detection. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 218–229. IEEE, 2019.
- [71] Alina Lazar, Sarah Ritchey, and Bonita Sharif. Generating duplicate bug datasets. In *Proceedings of the 11th working conference on mining software repositories*, pages 392–395, 2014.
- [72] Luiz Alberto Ferreira Gomes, Ricardo da Silva Torres, and Mario Lúcio Côrtes. Bug report severity level prediction in open source software: A survey and research opportunities. *Information and Software Technology*, 2019.
- [73] Jamal Uddin, Rozaida Ghazali, Mustafa Mat Deris, Rashid Naseem, and Habib Shah. A survey on bug prioritization. *Artificial Intelligence Review*, 47(2):145–180, 2017.
- [74] Zhengjie Xu, Tieke He, Weiqiang Zhang, Yabin Wang, Jia Liu, and Zhenyu Chen. Exploring the influence of time factor in bug report prioritization. In *SEKE*, pages 243–248, 2016.
- [75] Waheed Yousuf Ramay, Qasim Umer, Xu Cheng Yin, Chao Zhu, and Inam Illahi. Deep neural network-based severity prediction of bug reports. *IEEE Access*, 7:46846–46857, 2019.
- [76] Qasim Umer, Hui Liu, and Inam Illahi. Cnn-based automatic prioritization of bug reports. *IEEE Transactions on Reliability*, 2019.
- [77] Krishna Kumar Chaturvedi and VB Singh. Determining bug severity using machine learning techniques. In *2012 CSI Sixth International Conference on Software Engineering (CONSEG)*, pages 1–6. IEEE, 2012.
- [78] Meera Sharma, Punam Bedi, KK Chaturvedi, and VB Singh. Predicting the priority of a reported bug using machine learning techniques and cross project validation. In *2012 12th International Conference on Intelligent Systems Design and Applications (ISDA)*, pages 539–545. IEEE, 2012.
- [79] Rajni Jindal, Ruchika Malhotra, and Abha Jain. Prediction of defect severity by mining software project reports. *International Journal of System Assurance Engineering and Management*, 8(2):334–351, 2017.
- [80] Jaweria Kanwal and Onaiza Maqbool. Bug prioritization to facilitate bug report triage. *Journal of Computer Science and Technology*, 27(2):397–412, 2012.

- [81] Yuan Tian, David Lo, and Chengnian Sun. Information retrieval based nearest neighbor classification for fine-grained bug severity prediction. In *2012 19th Working Conference on Reverse Engineering*, pages 215–224. IEEE, 2012.
- [82] Mamdouh Alenezi and Shadi Banitaan. Bug reports prioritization: Which features and classifier to use? In *2013 12th International Conference on Machine Learning and Applications*, volume 2, pages 112–116. IEEE, 2013.
- [83] Yuan Tian, David Lo, and Chengnian Sun. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*, pages 200–209. IEEE, 2013.
- [84] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering*, 20(5):1354–1383, 2015.
- [85] Yang Feng, Zhenyu Chen, James A Jones, Chunrong Fang, and Baowen Xu. Test report prioritization to assist crowdsourced testing. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 225–236, 2015.
- [86] Yang Feng, James A Jones, Zhenyu Chen, and Chunrong Fang. Multi-objective test report prioritization using image understanding. In *2016 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 202–213. IEEE, 2016.
- [87] Shikai Guo, Rong Chen, and Hui Li. Using knowledge transfer and rough set to predict the severity of android test reports via text mining. *Symmetry*, 9(8):161, 2017.
- [88] Guofeng Gao, Hui Li, Rong Chen, Xin Ge, and Shikai Guo. Identification of high priority bug reports via integration method. In *CCF Conference on Big Data*, pages 336–349. Springer, 2018.
- [89] Shikai Guo, Rong Chen, Miaomiao Wei, Hui Li, and Yaqing Liu. Ensemble data reduction techniques and multi-rsmote via fuzzy integral for bug report classification. *IEEE Access*, 6:45934–45950, 2018.
- [90] Shikai Guo, Miaomiao Wei, Siwen Wang, Rong Chen, Chen Guo, Hui Li, and Tingting Li. Identify high-impact bug reports by combining the data reduction and imbalanced learning strategies. *Applied Sciences*, 9(18):3663, 2019.
- [91] Madhu Kumari and VB Singh. An improved classifier based on entropy and deep learning for bug priority prediction. In *International Conference on Intelligent Systems Design and Applications*, pages 571–580. Springer, 2018.
- [92] Qasim Umer, Hui Liu, and Yasir Sultan. Emotion based automated priority prediction for bug reports. *IEEE Access*, 6:35743–35752, 2018.
- [93] Han Wang, Min Zhou, Xi Cheng, Guang Chen, and Ming Gu. Which defect should be fixed first? semantic prioritization of static analysis report. In *International Conference on Software Analysis, Testing, and Evolution*, pages 3–19. Springer, 2018.
- [94] Shikai Guo, Rong Chen, Hui Li, Tianlun Zhang, and Yaqing Liu. Identify severity bug report with distribution imbalance by cr-smote and elm. *International Journal of Software Engineering and Knowledge Engineering*, 29(02):139–175, 2019.
- [95] Abeer Hamdy and Abdulrahman El-Laithy. Smote and feature selection for more effective bug severity prediction. *International Journal of Software Engineering and Knowledge Engineering*, 29(06):897–919, 2019.
- [96] Abeer Hamdy and AbdulRahman El-Laithy. Semantic categorization of software bug repositories for severity assignment automation. In *Integrating Research and Practice in Software Engineering*, pages 15–30. Springer, 2020.
- [97] M Irtaza Nawaz Tarar, Mubashir Ali, and Wasi Haider Butt. Bug report summarization: A systematic literature review. In *Proceedings of the 2019 11th International Conference on Education Technology and Computers*, pages 257–261, 2019.
- [98] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. Automatic summarization of bug reports. *IEEE Transactions on Software Engineering*, 40(4):366–380, 2014.
- [99] Nishant Jha and Anas Mahmoud. Using frame semantics for classifying and summarizing application store reviews. *Empirical Software Engineering*, 23(6):3734–3767, 2018.
- [100] Xiaochen Li, He Jiang, Dong Liu, Zhilei Ren, and Ge Li. Unsupervised deep bug report summarization. In *Proceedings of the 26th Conference on Program Comprehension*, pages 144–155, 2018.
- [101] Di Liu, Xiaofang Zhang, Yang Feng, and James A Jones. Generating descriptions for screenshots to assist crowdsourced testing. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 492–496. IEEE, 2018.
- [102] Shengcheng Yu. Crowdsourced report generation via bug screenshot understanding. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1277–1279. IEEE, 2019.
- [103] Xin Chen, He Jiang, Zhenyu Chen, Tieke He, and Liming Nie. Automatic test report augmentation to assist crowdsourced testing. *Frontiers of Computer Science*, 13(5):943–959, 2019.
- [104] Rui Hao, Yang Feng, James A Jones, Yuying Li, and Zhenyu Chen. Ctras: Crowdsourced test report aggregation and summarization. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, pages 900–911. IEEE, 2019.
- [105] Yuying Li, Rui Hao, Yang Feng, James A Jones, Xiaofang Zhang, and Zhenyu Chen. Ctras: a tool for aggregating and summarizing crowdsourced test reports. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 406–409, 2019.