

Images Don't Lie: Duplicate Crowdfunding Reports Detection With Screenshot Information

Junjie Wang^{1,3}, Mingyang Li^{1,3}, Song Wang⁴, Tim Menzies⁵, Qing Wang^{1,2,3,*}

¹Laboratory for Internet Software Technologies, ²State Key Laboratory of Computer Science, Institute of Software Chinese Academy of Sciences, Beijing, China

³University of Chinese Academy of Sciences, Beijing, China, *Corresponding author

⁴Electrical and Computer Engineering, University of Waterloo, Canada

⁵Department of Computer Science, North Carolina State University, Raleigh, NC, USA

Email: {wangjunjie, limingyang, wq}@itechs.iscas.ac.cn, song.wang@uwaterloo.ca, tim@menzies.us

Abstract

Context: Crowdfunding is effective especially when it comes to the feedback on GUI systems, or subjective opinions about features. Despite of this, we find crowdfunding reports are highly duplicated, i.e., 82% of them are duplicates of others. Most of the existing approaches mainly adopted textual information for duplicate detection, and suffered from low accuracy because of the lexical gap. Our observation on real industrial crowdfunding data found that when dealing with crowdfunding reports of GUI systems, the reports would be accompanied with images, i.e., the screenshots of the tested app. We assume the screenshot to be valuable for duplicate crowdfunding report detection because it reflects the real context of the bug and is not affected by the variety of natural languages.

Objective: We aim at automatically detecting duplicate crowdfunding reports that could help reduce triaging effort.

Method: In this work, we propose SETU which combines information from the Screenshots and the Textual descriptions to detect duplicate crowdfunding reports. We extract four types of features to characterize the screenshots (i.e., image structure feature and image color feature) and the textual descriptions (i.e., TF-IDF feature and word embedding feature), and design a hierarchical algorithm to detect duplicates based on the four similarity scores derived from the four features respectively.

Results: We investigate the effectiveness of SETU on 12 projects with 3,689 reports from one of the Chinese largest crowdfunding platforms. Results show that recall@1 achieved by SETU is 0.44 to 0.79, recall@5 is 0.66 to 0.92, and MAP is 0.21 to 0.58 across all experimental projects. Furthermore, SETU can outperform existing state-of-the-art approaches significantly and substantially.

Conclusion: Through combining the screenshots and textual descriptions, our proposed SETU can improve the duplicate crowdfunding reports detection performance.

Keywords: Crowdfunding, Duplicate report, Similarity detection

1. Introduction

Crowdfunding is an emerging trend in software testing which accelerates testing processes by attracting online crowd workers to accomplish various types of testing tasks [35, 12, 37, 13, 36, 15]. It entrusts testing tasks to crowd workers whose diverse testing environments/platforms, background, and skill sets could significantly contribute to more reliable, cost-effective, and efficient testing results.

The benefit of crowdfunding must be carefully assessed with respect to the cost of the technique. At first place, crowdfunding is a scalable testing method under which large software systems can be tested with appropriate results. This is particular true when the testing is related with the feedback on GUI systems, or subjective opinions about different features.

One aspect of crowdfunding which is not received enough attention in prior work is the confusion factors in crowdfunding results. Review 2-b Our observation on real industrial data shows that an average of 82% crowdfunding reports are duplicate

(see Table 3 for details), which suggests much of the crowdfunding work can be optimized. A significant problem with such a large number of duplicate reports is that the subsequent analysis by software testers becomes extremely complicated. For example, we find that merely working through 500 crowdfunding reports to find the duplicate ones takes almost the whole working day of a tester. This paper mostly removes that effort by a novel method for detection of duplicate reports.

The issue of duplicate reports has been studied in terms of textual descriptions [28, 38, 18, 30, 31, 29, 26, 33, 3, 44, 24, 4, 1, 17, 27, 42] (see details in Section 3). However, in practice, it is common that different people might use different terminologies, or write about different phenomena to describe the same issue [38, 24, 33, 42], which makes the descriptions often confusing. Because of this, most existing approaches for duplicate report detection suffer from low accuracy. However, when dealing with crowdfunding reports of GUI systems, besides the textual descriptions, often the feedback is in the form of images. Our observation on real industrial crowdfunding data re-

veals that an average of 94% crowdtesting reports are accompanied with an image, i.e., screenshot of the app. We suppose this is another valuable source of information for detecting duplicate crowdtesting reports. Compared with the textual description, a screenshot can reflect the real context of the bug and is not affected by the variety of natural languages.

In this paper, we propose SETU which combines information from the ScrEenshots and the TextUal descriptions to detect duplicate crowdtesting reports. We first extract two types of features from screenshots (i.e., image structure feature and image color feature), and two types of features from textual descriptions (i.e., TF-IDF feature and word embedding feature). We then obtain the screenshot similarity and textual similarity through computing the similarity scores based on the four types of features. To decide the duplicates of a query report, SETU adopts a hierarchical algorithm. Specifically, if the screenshot similarity between the query report and candidate report is higher than a threshold, we treat the candidate report as the first class and rank all reports in the first class by their textual similarity. Otherwise, we treat it as the second class (follow behind the first class) and rank all reports in this class by their combined textual similarity and screenshot similarity. Finally, we return a list of candidate duplicate reports of the query report, with the ranked reports of the first class followed by the ranked reports of the second class.

We experimentally evaluate the effectiveness of SETU on 12 projects with 3,689 crowdtesting reports from one of the Chinese largest crowdtesting platforms. Results show that the recall@1 achieved by SETU is 0.44 to 0.79, recall@5 is 0.66 to 0.92, and MAP is 0.21 to 0.58 across all experimental projects.

Review 1-a This denotes, for a new-coming crowdtesting report, in 44% - 79% of the time, SETU can correctly identify its duplicate reports with only a single recommendation. And in 66% - 92% of the time, SETU can correctly identify its duplicate reports with five recommendations. These results significantly and substantially outperform three state-of-the-art and typical duplicate detection approaches. In addition, we also experimentally evaluate the necessity of screenshots and textual descriptions in duplicate detection, as well as the relative effect of the four types of features.

This paper makes the following contributions:

- We show that the screenshots are valuable in duplicate crowdtesting reports detection and we further demonstrate the need to use both the screenshots and the textual descriptions in detecting duplicate crowdtesting reports.
- We propose a novel approach (SETU) for duplicate crowdtesting report detection, which combines the information from screenshots and textual descriptions hierarchically.
- We evaluate the effectiveness of SETU on 12 projects from one of the Chinese largest crowdtesting platforms, and results are promising.

The rest of this paper is organized as follows: Section 2 describes the background and motivation of this study, while Section 3 surveys related work. Section 4 presents our proposed

approach of duplicate detection. Section 5 and 6 show the experimental setup and evaluation results respectively. Section 7 provides a detailed discussion and threats to validity. Finally, we summarize this paper in Section 8.

2. Background and Motivation

2.1. Background

In this section, we present a brief background of crowdtesting to help better understand the challenges we meet in real industrial crowdtesting practice.

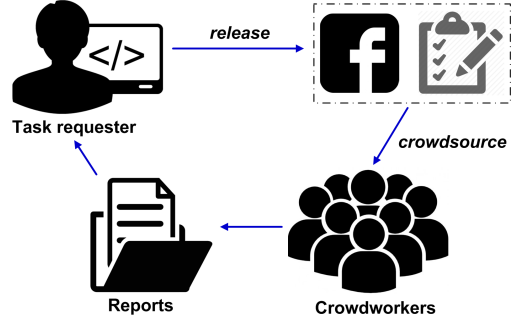


Figure 1: Procedure of crowdtesting

Our experiment is conducted with Baidu CrowdTest crowdtesting platform¹. As shown in Figure 1, in general, the task requester² prepares the software under test and testing tasks, and distributes them on the crowdtesting platform. Then, the crowd workers can sign in to conduct the tasks and are required to submit the crowdtesting reports, i.e., to describe the process and result of the testing task s/he carried on, which include the input, operation steps, result description, and screenshot. Table 1 demonstrates an example of the crowdtesting report³.

In order to attract more workers, testing tasks are often financially compensated. Under this context, workers can submit hundreds of reports for a crowdtesting task. This platform delivers approximately 100 projects per month, and receives more than 1,000 test reports per day on average. **Review 1-b** An observation on our experimental dataset shows that an average of 82% crowdtesting reports are duplicates of other reports (see Table 3 for details).

Currently in this platform, the testers need to manually inspect these crowdtesting reports to identify the duplicate ones. However, inspecting 500 reports manually could take almost the whole working day of a tester. Obviously, such process is time-consuming and low-efficient.

Review 1-c Most of the existing approaches mainly adopted textual information for duplicate detection, and suffered from

¹Baidu (baidu.com) is the largest Chinese search service provider. Its crowdtesting platform (test.baidu.com) is also one of the largest crowdtesting platforms in China.

²The task requesters are usually the testers in practice, so we also call them as the testers in the following paper.

³Note that, since all the reports are written in Chinese in our experiment projects, we translate them into English to facilitate understanding.

Table 1: An example of crowdtesting report

Attribute	Description: <i>example</i>
Environment	Phone type: <i>Samsung SN9009</i> Operating system: <i>Android 4.4.2</i> ROM information: <i>KOT49H.N9009</i> Network environment: <i>WIFI</i>
Crowd worker	Id: <i>123456</i> Location: <i>Beijing Haidian District</i>
Testing task	Id: <i>01</i> Name: <i>Incognito mode</i>
Input and operation steps	<i>Input "sina.com.cn" in the browser, then click the first news. Select "Setting" and then set "Incognito Mode". Click the second news in the website. Select "Setting" and then select "History".</i>
Result description	<i>"Incognito Mode" does not work as expected. The first news, which should be recorded, does not appear in "History".</i>
Screenshot	
Assessment	Passed or failed given by crowd worker: <i>Failed</i>

low accuracy because of the lexical gap in natural language. This is especially true for crowdtesting because the reports are written by crowdworkers who are usually unprofessional in software testing. Therefore, we propose to include screenshots in duplicate detection to help address the issue. Section 2.2 will provide two motivating examples to show how textual descriptions lead to confusing understanding, and how screenshots and textual descriptions can complement each other in duplicate detection.

2.2. Motivation

In this section, we present two examples from Baidu CrowdTest crowdtesting platform to motivate the need of using both the screenshots and the textual descriptions in duplicate crowdtesting report detection. These examples draw from a sport application, i.e., JIAJIA Sport. It can automatically record and analyze users' sport-related information such as running, rope skipping, etc. To better test this application, its testers distribute the testing task on Baidu CrowdTest crowdtesting platform, and 462 crowdtesting reports are submitted by the crowd workers. By analyzing these crowdtesting reports, we find the following two motivating examples.

2.2.1. Motivating Example 1: Descriptions Could be Confusing

Crowdtesting reports *Rope-145* and *Rope-270* are about the sharing function. Their descriptions are as follows:

Rope-145: *I press the qzone⁴ sharing button in the bottom and want to share my rope skipping record, but nothing happens.*

Rope-270: *I press the sharing button and want to share my rope skipping record to qzone, but nothing happens except a failure notice.*

Both descriptions contain such words as "sharing button", "rope skipping record", and "qzone". Using traditional duplicate detection approaches, these two reports would be identified as duplicates with a high probability. However, if the screenshot information (see Figure 2) is considered, they can be easily determined as non-duplicates, which is the ground truth. The screenshot of *Rope-145* is about the billboard of rope skipping record, and the crowdtesting report reveals a bug about sharing



Figure 2: Motivating example 1

the ranking of rope skipping record. For the screenshot of *Rope-270*, it demonstrates the detail page of rope skipping record, and the report reveals a bug about sharing the detailed record. In this sense, the screenshot provides the context-related information and can help better detect duplicate reports. One may argue that the above information should be in the *operation steps* (see Table 1) submitted by the crowd workers. However, the crowd workers are far from professional testers, and in our datasets only few reports contain the detailed and correct operation steps.

Finding 1: The textual descriptions of crowdtesting reports may easily lead to confusing understanding. With the help of context-related information provided by screenshots, the duplicate crowdtesting reports can be detected more accurately.

2.2.2. Motivating Example 2: Screenshots Usually Lack Details

In this example, crowdtesting reports *Rope-62* and *Rope-217* have similar screenshots (see Figure 3). If the duplicate detection is only based on the screenshot information, these two reports would be determined as duplicates with a high probability. However, the ground truth is just the opposite. Their descriptions are as follows:

Rope-62: *I walked for 10 minutes, but the steps only increased by 10. An hour later, i just sat on my chair, but the steps increased sharply.*

Rope-217: *In the detail page, there is indeed step record for today, but there is no step record for this week.*

From the descriptions, we can easily observe that these two crowdtesting reports involve two different bugs, although under the same function which is denoted by the two highly similar screenshots. In this sense, a screenshot merely demonstrates the context-related information about a crowdtesting report. We still need to refer to the textual descriptions to finally determine whether they are duplicates.

⁴qzone is a popular social networking website in China.

Table 2: Summary of duplicate bug report detection researches

Article	Basic idea	Experiment projects	Performance	Baseline	Publication year
[28] Detection of Duplicate Defect Reports Using Natural Language Processing	Use natural language processing technique (i.e., tokenization, stemming, stop words removal, vector space representation) to compute the similarity	Sony Ericsson Mobile communications	Recall@5 = 30%, Recall@10 = 38%, Recall@15 = 42%	N/A	ICSE'2007
[38] An Approach to Detecting Duplicate Bug Reports using Natural Language and Execution Information	Use both natural language information and execution information to compute the similarity, and combine the two similarity values based on heuristics	Firefox	Recall@1 = 67%, Recall@10 = 93%	[28]	ICSE'2008
[18] Automated Duplicate Detection for Bug Tracking Systems	Build a machine learning classifier which combines the surface features of the report, textual similarity metrics, and graph clustering algorithms	Mozilla	Recall@1 = 25%, Recall@5 = 38%, Recall@10 = 45%	[28]	DSN'2008
[30] A Discriminative Model Approach for Accurate Duplicate Bug Report Retrieval	Extract 54 features to measure the textual similarity of a pair of reports based on the term weighting, then build a machine learning classifier on these features	Firefox, Eclipse, OpenOffice	Recall@1 = 32% - 38%, Recall@5 = 48% - 52%, Recall@10 = 56% - 61%	[28], [38], [18]	ICSE'2010
[31] Detecting Duplicate Bug Report Using Character N-Gram-Based Features	Compute the semantic and lexical similarity based on the character-level n-gram model	Eclipse	Recall@10 = 40%, Recall@20 = 48%, Recall@50 = 61%	N/A	APSEC'2010
[29] Towards More Accurate Retrieval of Duplicate Bug Reports	Use both the textual information and other fields information (e.g., product, component, versions) to measure the similarity, extend BM25F (an effective similarity formula) to handle lengthy structured report by considering weights of terms	Eclipse, OpenOffice, Mozilla	Recall@1 = 37% - 42%, Recall@5 = 58% - 62%, Recall@10 = 63% - 69%, MAP = 45% - 53%	[30], [31]	ASE'2011
[26] Detecting Bug Duplicate Reports through Local Reference	Find the reports whose submit time is within the sliding time window, then rank these reports based on TF-IDF	Firefox	Recall@1 = 20%, Recall@5 = 36%, Recall@10 = 44%	N/A	PROMISE'2011
[33] Improved Duplicate Bug Report Identification	Use extended BM25F [29] for similarity measurement, introduce relative similarity by considering the top-k most similar reports (rather than only considering top-1 most similar report)	Mozilla	Recall@4 = 24%, Recall@20 = 25%	[18]	CSMR'2012
[3] Automated Duplicate Bug Report Classification using Subsequence Matching	Hypothesize that two reports that have the longest ordered sequence of common words are more likely to be duplicates than those have the same frequency of unordered words; Propose common sequence matching approach	Firefox	Recall@1 = 30%, Recall@5 = 51%, Recall@10 = 58%	N/A	HASE'2012
[44] Learning to Rank Duplicate Bug Reports	Identify 9 textual and statistical features of bug reports, create a ranking model based on these features to rank the duplicate bugs higher, learn the weights of the features by applying the stochastic gradient descent algorithm	Eclipse	Recall@1 = 49%, Recall@5 = 69%, Recall@10 = 76%, MRR = 58%	[30], [29]	CIKM'2012
[24] Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling	Use information retrieval and topic modeling techniques to measure the textual similarity	Eclipse, OpenOffice, Mozilla	Recall@1 = 42% - 57%, Recall@5 = 66% - 76%, recall@10 = 80% - 86%	[29]	ASE'2012
[4] A Fusion Approach For Classifying Duplicate Problem Reports	Use multi-label classification to assign each report with multiple duplicate prediction results and use a fusion role to combine the results	Firefox	recall@1 = 31%, recall@5 = 50%, recall@10 = 58%	[30]	ISSRE'2013
[1][17] A Contextual Approach Towards More Accurate Duplicate Bug Report Detection and Ranking	Utilize contextual information about software-quality attributes, software-architecture terms, and system-development topics when computing the similarity	Eclipse, OpenOffice, Mozilla, Android	MAP = 29% - 51%	[29]	MSR'2013, EMSE'2016
[27] An Empirical Study on Recommendations of Similar Bugs	Use bug component field and textual similarity of bug descriptions to measure the similarity	Mozilla	Recall@1 = 36%, Recall@3 = 43%	[29]	SANER'2016
[42] Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports	Apply information retrieval and word embedding technique to measure the textual similarity of bug titles and descriptions, and consider bug product and component fields in the final similarity of two reports	Eclipse, Mozilla	Recall@1 = 16% - 19%, Recall@5 = 37% - 43%, Recall@10 = 48% - 53%, MAP = 26% - 33%, MRR = 48% - 59%	[27]	ISSRE'2016

Note that, [17] is the extension of [1], so we put them together and present the results of the extended paper [17].

Finding 2: Screenshots of crowdtesting reports mainly demonstrate the context-based information. Under a specific context, only with the detailed illustration provided by the textual description, the duplicated reports can be accurately detected.

3. Related Work

3.1. Crowdtesting

Crowdtesting has been applied to facilitate many testing tasks. Chen and Kim [7] applied crowdtesting to test case generation. They investigated object mutation and constraint solving issues underlying existing test generation tools, and presented a puzzle-based automatic testing environment. Musson

et al. [23] proposed an approach, in which the crowdworker was used to measure real-world performance of software products. Gomide et al. [16] proposed an approach that employed a deterministic automata to help usability testing. Adams et al. [14] proposed MoTIF to detect and reproduce crashes in mobile apps after their deployment in the wild.

These studies leverage crowdtesting to solve the problems in traditional testing activities, while some other approaches focus on the new encountered problem in crowdtesting.

Feng et al. [12, 13] proposed approaches to prioritize test reports in crowdtesting. They designed strategies to dynamically select the most risky and diversified test report for inspection in each iteration. Jiang et al. [19] proposed the test report fuzzy clustering framework by aggregating redundant and multi-bug crowdtesting reports into clusters to reduce the number of in-

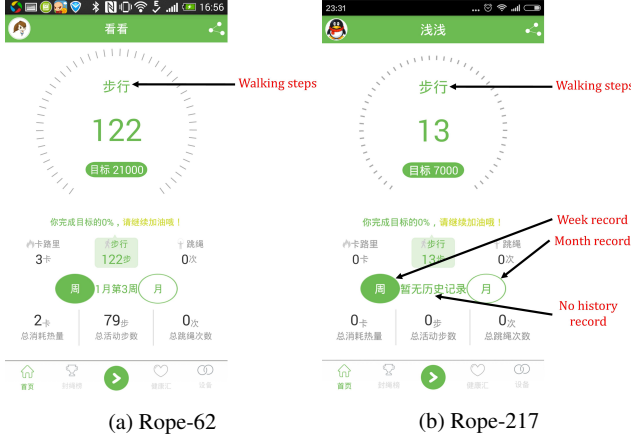


Figure 3: Motivating example 2

spected test reports. Wang et al. [35, 37, 36] proposed approaches to automatically classify crowdtesting reports. Their approaches can overcome the different data distribution among different software domains, and attain good classification results. Liu et al. [20] proposed an automatic approach to generate descriptive words for the screenshots based on the language model and Spatial Pyramid Matching technique. Cui et al. [11, 10] and Xie et al. [40] proposed crowd worker selection approaches to recommend appropriate crowd workers for specific crowdtesting tasks. These approaches considered the worker’s experience, relevance with the task, diversity of testing context, etc., and recommend a set of workers who can detect more bugs.

In this work, we focus on detecting the duplicated crowdtesting reports to facilitate real industrial crowdtesting practice.

3.2. Duplicated Bug Report Detection

Many approaches have been proposed to detect duplicate bug reports [28, 38, 18, 30, 31, 29, 26, 33, 3, 44, 24, 4, 1, 17, 27, 42]. The main focus of duplicate report detection is to obtain the similarity of two reports. Runeson et al. [28] proposed the first duplicate report detection approach which uses natural language information of bug reports to compute the similarity. Later, other sources of information were utilized in similarity measurement, e.g., execution trace [38], fields information as product and component [33, 27, 42]. Meanwhile, techniques to improve the similarity measurement precision were also introduced, e.g., BM25F similarity measurement [29], topic modeling [24], word embedding technique [42], etc.

Table 2 presents a summary of existing duplicate bug report detection researches.

These approaches mainly leverage the textual information and fields information to detect the duplicate reports. In this work, we introduce the screenshot information to facilitate the duplicate crowdtesting report detection. Our proposed approach combines the information from both the screenshots and the textual descriptions, and can detect duplicate crowdtesting reports with high accuracy.

To evaluate the effectiveness of our proposed approach (in Section 5 and 6), we choose three state-of-the-art and typical approaches [24, 27, 42] as the baselines. The reason why we use [27, 42] is because they are the latest two researches which can be treated as the state-of-the-art approaches. The reason why we use [24] is because they can achieve the highest performance across all existing researches. We had planned to include [17], which is also one of the latest researches, as baseline. However, the contextual words utilized in that work are English-oriented; simply translating these words to Chinese or translating our crowdtesting reports to English would bring serious information loss. Therefore, this paper utilize three state-of-the-art and typical approaches [24, 27, 42] as baseline in evaluation (see details in Section 5.3).

4. Approach

Motivated by the two examples in Section 2.2, we propose a duplicate detection approach (SETU), which combines information from both the Screenshots and the Textual descriptions to detect duplicate crowdtesting reports. Figure 4 illustrates the overview of SETU.

Our approach is organized as a pipeline comprising three phases: feature extraction, similarity calculation, and duplicate detection.

In the feature extraction phase, for the screenshots, we extract two types of features, i.e., image structure feature and image color feature (details are in Section 4.1). For the textual descriptions, we also extract two types of features, i.e., TF-IDF (Term Frequency and Inverse Document Frequency) feature and word embedding feature (details are in Section 4.2).

In the similarity calculation phase, based on the four types of features, we compute four similarity scores between the query report and each of the pending reports, and obtain the screenshot similarity and textual similarity. Cosine similarity, which is commonly used for measuring distance [35, 28, 27, 39], is employed in our approach.

In the duplicate detection phase (details are in Section 4.3), we design a hierarchical algorithm. In detail, if the screenshot similarity between the query report and pending report is larger than a specific threshold, we treat the pending report as first class and rank all reports in the first class by their textual similarity. Otherwise, we treat it as second class (follow behind the first class) and rank all reports in this class by their combined textual similarity and screenshot similarity. Finally, we return a list of candidate duplicate reports of the query report, with the ranked reports of the first class followed by the ranked reports of the second class. The reason why we separate reports in two classes is to take the advantages of the information provided by screenshots. As described in Section 2.2, the screenshots can provide the context-related information, and reports with different screenshots are very unlikely to be duplicate with each other. We will further discuss the performance of other combinations of the screenshot similarity and textual similarity in Section 7.4.

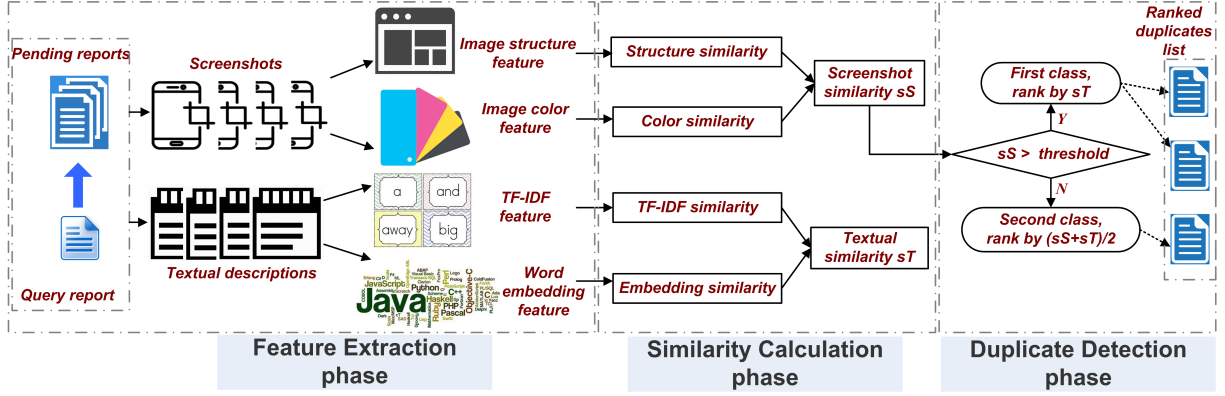


Figure 4: Overview of SETU

4.1. Extracting Screenshot Features

We extract screenshot features from the image of screenshot accompanied with each crowdtesting report. Note that, in our experimental projects, all the crowdtesting reports contain zero or one screenshot (with details in Section 5.2). For reports which do not have a screenshot, we use a default blank picture to extract the features. Future work will consider the situation that one report contains several screenshots. We use the following two types of features.

4.1.1. Image Structure Feature

The geometric structure of an image exhibits fundamental information for distinguishing screenshots [8, 34]. Images with highly similar geometric structures (e.g., line segments) would be probably the same screenshot.

Review 3-a Gist descriptor [25] can capture the spatial structure of an image through the segmentation and processing of individual regions. Its general idea is as follows: propose a set of perceptual dimensions (naturalness, openness, roughness, expansion, ruggedness) to represent the dominant spatial structure of an image; estimate these dimensions based on the spectral and coarsely localized information; generate a multidimensional space in which scenes sharing membership in semantic categories are projected closed together.

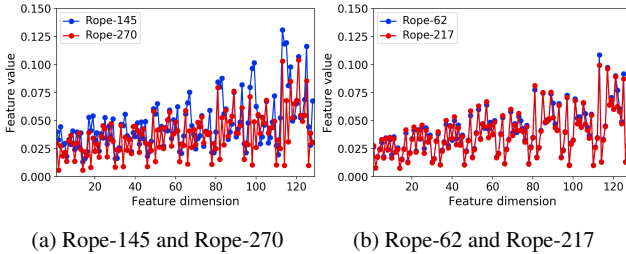


Figure 5: Image structure features

We use the publicly available package⁵ with default parameters to extract image structure feature. It results in a 128-dimensional image structure feature vector for each image.

⁵<http://people.csail.mit.edu/torralba/code/spatialenvelope/>

Figure 5 shows the image structure features for the four screenshots in Figure 2 and Figure 3. **Review 3-c** The x axis is the feature’s dimension (i.e., 128), while the y axis is the value of the feature in each dimension. We can easily find that the image structure feature vectors of *Rope-145* and *Rope-270* are obviously different with each other, while the structure feature vectors of *Rope-62* and *Rope-217* are almost the same. This coincides with our visual perception.

4.1.2. Image Color Feature

Color is another basic indicator of visual contents, and is often used to describe and represent an image [8, 34]. Images with highly similar color distributions are probably the same screenshot.

Review 3-b MPEG-7 descriptor [21] can capture the representative colors on a grid superimposed of an image based on segmentation and clustering. Its general idea is as follows: use the edgeflow algorithm to segment image; perform color clustering on each segmented region to obtain its representative colors and calculate the percentage of these colors; each representative color and its corresponding percentage form a pair of attributes that describe the color characteristics in an image.

We also use the publicly available software⁶ to extract the image color feature. It results in a 189-dimensional image color feature vector for each image.

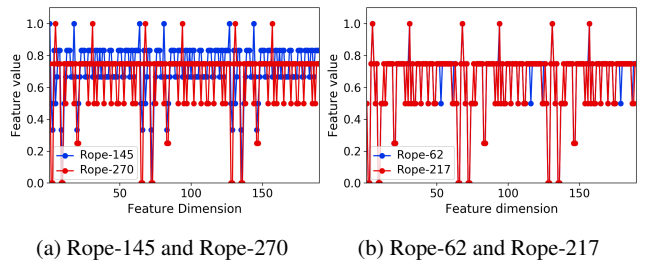


Figure 6: Image color features

Figure 6 demonstrates the image color feature vectors for the four screenshots in Figure 2 and Figure 3. **Review 3-d** The

⁶<http://www.semanticmetadata.net/>

x axis is the feature's dimension (i.e., 189), while the y axis is the value of the feature in each dimension. The image color feature vector of *Rope-145* exerts obvious difference with the vector of *Rope-270*, while the color vector of *Rope-62* is quite similar with the vector of *Rope-217*. Just as the image structure feature, this coincides with our visual perception.

Both structure feature and color feature are widely used in image processing tasks [8, 34], so we adopt both of them in the duplicate crowdtesting report detection and explore their performance in Section 6.

4.2. Extracting Textual Features

We extract textual features from the textual descriptions of crowdtesting reports.

We first collect different sources of textual descriptions together (input, operation steps, and result description), and then conduct the natural language processing to remove noise and extract core terms. Specifically, because the crowdtesting reports in our experiment are written in Chinese, we adopt ICTCLAS⁷ for word segmentation, and segment descriptions into words. We then remove **stopwords** (i.e., “am”, “on”, “the”, etc.) to reduce noise. Note that, crowd workers often use different words to express the same concept, so we introduce the **synonym replacement** technique to mitigate this problem. Synonym library of LTP⁸ is adopted. The remaining terms are saved and will be used to extract the following two types of features.

4.2.1. TF-IDF Feature

TF-IDF (Term Frequency and Inverse Document Frequency) is one of the most popular feature for representing textual documents in information retrieval. The main idea of TF-IDF is that if a term appears many times in one report and a few times in the other report, the term has a good capability to differentiate the reports, and thus the term has high TF-IDF value. Specifically, given a term t and a report r , $TF(t, r)$ is the number of times that term t occurs in report r , while $IDF(t)$ is obtained by dividing the total number of reports by the number of reports containing term t . TF-IDF is computed as: $TF - IDF(t, r) = TF(t, r) \times IDF(t)$.

With the above formula, the textual description of a report r can be represented as a TF-IDF vector, i.e., $r = (w_1, w_2, \dots, w_n)$, where w_i denotes the TF-IDF value of the i^{th} terms in report r .

4.2.2. Word Embedding Feature

Word embedding is a feature learning technique in natural language processing where individual words are no longer treated as unique symbols, but represented as d -dimensional vector of real numbers that capture their contextual semantic meanings [22, 5].

We use the publicly available software⁹ to obtain the word embedding of a report. With the trained word embedding

model, each word can be transformed into a d -dimensional vector where d is set to 100 as suggested in previous studies [42, 41]. Meanwhile a crowdtesting report can be transformed into a matrix in which each row represents a term in the report. We then transform the report matrix into a vector by averaging all the word vectors the report contains as previous work did [42]. Specifically, given a report matrix that has n rows in total, we denote the i^{th} row of the matrix as r_i and the transformed report vector v_d is generated as follows:

$$v_d = \frac{\sum_i r_i}{n} \quad (1)$$

With the above formula, each crowdtesting report can be represented as a word embedding vector.

The TF-IDF feature focuses on the similarity of reports considering the term matching, while the word embedding feature concerns more on the relationship of terms considering the context they appear. We adopt both of them in our approach and investigate their performance in duplicate report detection in Section 6.

4.3. Conducting Duplicate Report Detection

Following the previous studies [28, 38, 18, 30, 31, 29, 26, 33, 3, 44, 24, 4, 1, 17, 27, 42], our duplicate crowdtesting report detection problem is formulated as follows: Given a query report of a crowdtesting project, our approach would recommend a list of duplicate reports from all the pending reports of the project and rank them by their probabilities to be duplicates. We design a hierarchical algorithm (Algorithm 1) to detect the duplicate reports.

Algorithm 1 Duplicate report detection algorithm

Input:

Pending crowdtesting report set R ;
Query report q ; Threshold $thres$

Output:

A list of duplicate reports D ;

```

1: for each report  $r$  in  $R$  and  $q$  do
2:   Extract the image structure feature vector and image color feature vector from its screenshot;
3:   Extract the TF-IDF feature vector and word embedding feature vector from its textual description;
4: end for
5: for each report  $r$  in  $R$  do
6:   for  $fType$  in  $\{structure, color, tfidf, embedding\}$  do
7:     Compute the cosine similarity between the  $fType$  vector of  $q$  and  $r$ , and denote as  $s\_fType$ ;
8:   end for
9:    $s\_screenshot = (s\_structure + s\_color)/2$ ;
10:   $s\_textual = (s\_tfidf + s\_embedding)/2$ ;
11:   $s\_total = (s\_screenshot + s\_textual)/2$ 
12:  if  $s\_screenshot > thres$  then
13:    put  $r$  in  $D\_first$ 
14:  else
15:    put  $r$  in  $D\_second$ 
16:  end if
17: end for
18: Rank  $D\_first$  based on reports'  $s\_textual$  and put them in  $D$  sequentially
19: Rank  $D\_second$  based on reports'  $s\_total$  and put them in  $D$  sequentially (follow behind the reports of  $D\_first$ )

```

⁷ICTCLAS (<http://ictclas.nlpir.org/>) is widely used Chinese NLP platform.

⁸LTP (<http://www.ltp-cloud.com/>) is considered as one of the best cloud-based Chinese NLP platforms.

⁹<https://code.google.com/archive/p/word2vec/>

In the algorithm, the screenshot similarity can be seen as a filter because the screenshot provides the context-related information (see Section 2.2 for details). If two crowdtesting reports have different screenshots, they are unlikely to be duplicate even if the textual similarity between them is high. In addition, if two crowdtesting reports are accompanied with the same screenshot, whether they are duplicate reports mainly depends on the similarity of their textual descriptions.

The threshold to determine whether two screenshots are the same one is an input parameter. We explore the influence of this parameter on the detection performance in Section 5.6.

We have also experimented with different weights for $s_structure$ and s_color when combining them to obtain $s_screenshot$ (Line 9 in Algorithm 1), the weights for s_tfidf and $s_embedding$ when combining them to get $s_textual$ (Line 10 in Algorithm 1), as well as the weights for $s_screenshot$ and $s_textual$ when combining them to obtain s_total (Line 11 in Algorithm 1). Results turned out that when the two similarities have an equal weight, SETU can achieve a relative good and stable performance. Due to space limit, we do not present the detailed results. Note that, this does not imply the screenshot and textual descriptions are equally important, because the weights for $s_screenshot$ and $s_textual$ are only used in the second class of reports (Line 10 and 19 in Algorithm 1). Another note is that, for the reports in the second class, we have experimented with other ranking manners, i.e., by $s_screenshot$, by $s_textual$, and results turned out that with the ranking manner shown above, the detection performance is relative good and stable.

5. Experiment Design

5.1. Research Questions

Our evaluation addresses the following research questions:

- **RQ1 Effectiveness:** How effective is SETU in detecting duplicate crowdtesting reports?

RQ1 aims at evaluating the effectiveness of SETU in duplicate reports detection. We also compare SETU with the state-of-the-art approaches (see details in Section 5.3) to investigate whether and to what extent it improves over prior work.

- **RQ2 Necessity:** Are both screenshots and textual descriptions necessary in detecting duplicate crowdtesting reports?

This paper proposes to utilize both screenshots and textual descriptions in duplicate detection. RQ2 is to investigate whether both of them are necessary. We employ two additional experiments to investigate it. See details in Section 5.4.

- **RQ3 Replaceability:** What is the relative effect of the four types of features (i.e., TF-IDF, word embedding, image color, and image structure) in detecting duplicate crowdtesting reports?

SETU employs four types of features to characterize the screenshots (i.e., image color and structure) and the textual descriptions (i.e., TF-IDF and word embedding). RQ3 is to investigate the relative effect of these features. We use another four experiments for investigating this RQ, with details in Section 5.4.

5.2. Experimental Dataset

We mentioned that our experiment is based on crowdtesting reports from the repositories of Baidu CrowdTest crowdtesting platform. We collect all crowdtesting projects closed between June 1st 2017 and June 10th 2017. There are totally 12 crowdtesting projects.

Table 3 presents the detailed information of the projects with the application domain, the number of reports (i.e., $\#report$), the number and percentage of reports which have screenshots (i.e., $\#screenshots$ and $\%screenshots$).

There is a label accompanied with each report. It signifies a specific type of bug assigned by the tester in the company. Reports with the same label denote they are duplicates of each other. In this sense, we treat the pair of reports with the same label as duplicates, while the pair of reports with different labels as non-duplicates.

Review 2-c Table 3 also presents the percentage of duplicates (i.e., $\%duplicates$) and percentage of duplicate pairs (i.e., $\%duplicate\ pairs$).

$$\%duplicates = \frac{\#duplicates}{\#reports} = \frac{\#reports - \#unique\ labels\ for\ reports}{\#reports} \quad (2)$$

$$\%duplicate\ pairs = \frac{\#duplicate\ pairs}{\#total\ pairs} = \frac{\sum \#pairs\ of\ reports\ with\ same\ label}{\#pairs\ of\ all\ reports} \quad (3)$$

$$pairs\ of\ reports = \frac{\#reports * (\#reports - 1)}{2} \quad (4)$$

To verify the validity of these stored labels, we additionally conduct the random sampling and relabeling. In detail, we randomly select 4 projects, and sample 30% of crowdtesting reports from each selected project. A tester from the company is asked to relabel the duplicate reports, without knowing the stored labels. We then compare the difference between the stored duplicate results and the new labeled duplicate results. The percentage of difference for each project is all below 4%. Therefore, we believe the ground truth labels are relatively trustworthy.

For **training the word embedding model**, we use another textual dataset. In detail, we crawl the textual description of crowdtesting reports and task requirements of 500 crowdtesting projects from the experimental platform. The reason why we use this dataset is that previous studies have revealed that to train an effective word embedding model, a domain-specific dataset with large size is preferred [42, 41]. The size of our training dataset is 520M.

Table 3: Projects under investigation

	Domain	# report	# screenshot	% screenshot	# duplicates	% duplicates	total pairs	# duplicate pairs	% duplicate pairs
P1	Music	213	188	88%	208	97%	22578	7187	31%
P2	Weather	215	200	93%	168	78%	22578	1549	7%
P3	Beauty	230	216	94%	214	93%	26335	5682	22%
P4	News	243	236	97%	210	86%	29403	5203	18%
P5	Browser	252	237	95%	190	75%	31626	4347	14%
P6	Medical	271	255	94%	207	76%	36585	1165	3%
P7	Safety	282	270	96%	249	87%	40186	9358	23%
P8	Education	284	278	98%	233	82%	40186	1753	4%
P9	Health	317	307	97%	246	77%	50086	1344	3%
P10	Language	344	317	97%	236	68%	58996	1064	2%
P11	Sport	462	425	93%	391	84%	106491	2381	2%
P12	Efficiency	576	547	95%	490	85%	165600	17261	11%
	Summary	3,689		94%		82%			12%

5.3. Baselines

To explore the performance of our proposed SETU, we compare it with three state-of-the-art and typical baseline approaches. Note that, since there is no approach designed for duplicate *crowdtesting report* detection, we choose the approaches for duplicate *bug report* detection as our baselines. Section 3.2 has presented why we choose these three baselines.

Information retrieval with word embedding (IR-EM) [42]: It is the state-of-the-art technique for duplicate bug report detection. This approach first builds TF-IDF vector and word embedding vector and calculates two similarity scores based on them respectively. Meanwhile, it calculates a third similarity score based on bug product field and component field. Finally, it combines the three similarity scores into one final score and makes similar bug recommendation with it.

Similarity based on bug components and descriptions (NextBug) [27]: It is another state-of-the-art similarity-based approach for duplicate bug report detection. This approach first checks whether two reports have the same bug component field, if yes, processes the reports with standard information retrieval technique, calculates the cosine similarity of the reports, and ranks the reports with the similarity value.

Note that, for these two baselines, because the crowdtesting reports do not have the product or component fields, we use the most similar field, i.e., *test task id* for substitution.

Information retrieval with topic modeling (DBTM) [24]: It is the commonly-used technique for detecting duplicate bug reports. DBTM supposes a report as a textual document describing one or more technical issues, and duplicate reports as the documents describing the same technical issues. It then utilizes term-based and topic-based features to detect duplicates.

5.4. Experimental Setup

This section illustrates the experimental setup for answering each research question.

For answering RQ1, we compare SETU with three state-of-the-art approaches (see details in Section 5.3) to investigate whether and to what extent it improves over prior work.

For answering RQ2, we employ two additional experiments, i.e., *onlyText* and *onlyImage*, to investigate whether both screenshots and textual descriptions are necessary in duplicate reports detection. In detail, *onlyText* denotes ranking the reports only based on the textual similarity, while *onlyImage* denotes ranking the reports only based on the screenshot similarity.

For answering RQ3, we use another four experiments, i.e., *noTF*, *noEmb*, *noClr*, and *noStrc*, to investigate the relative effect of the four types of features utilized in duplicate detection. Each experiment denotes conducting the duplicate detection by removing one specific type of feature. For example, *noTF* denotes applying other three features except TF-IDF (i.e., only use word embedding, image color and image structure feature) for duplicate detection.

For all these experiments, we employ the commonly-used leave-one-out cross validation [6]. In detail, we use one crowdtesting project as the testing dataset to evaluate the performance of duplicate detection, and use the remaining crowdtesting projects as the training dataset to determine the optimal parameter value (see detail in Section 5.6).

We ran our experiments on a 3.40 GHz Intel Core i7-3770 machine with 8 GB of RAM. [Review 3-e](#) For the time-cost of feature extraction, extracting the image structure features of 188 screenshots (i.e., P1 in Table 3) consumes 132 seconds and extracting the image color features of 188 screenshots consumes 214 seconds. For training the word embedding model, it consumes 540 seconds. Extracting the TF-IDF features and word embedding features of 213 reports (i.e., P1 in Table 3) all consume less than 1 second.

5.5. Evaluation Metrics

We use three evaluation metrics, i.e., recall@k, mean average precision (MAP), and mean reciprocal rank (MRR), to evaluate the performance of duplicate detection. These metrics are commonly-used to evaluate the duplicate detection approaches (see Table 2 for details).

Given a query report q , its ground truth duplicate reports set $G(q)$, and the top-k recommended duplicate reports list produced by duplicate detection approach $R(q)$.

[Review 2-d](#) **Recall@k** checks whether a top-k recommendation is useful [24, 30, 42]. The definition of *recall@k* for a query report q is as follows:

$$recall@k = \begin{cases} 1, & \text{if } G(q) \cap R(q) \neq \emptyset \\ 0, & \text{Otherwise} \end{cases} \quad (5)$$

According to the formula, if there is at least one ground truth duplicate report in the top-k recommendation, the top-k recommendation is useful for the query report q . Given a set of query reports, we compute the proportion of useful top-k recommendations by averaging the *recall@k* of all query reports to get an

overall *recall@k*. As previous approaches, we set *k* as 1, 5, and 10 to obtain the performance.

MAP (Mean Average Precision) is defined as the mean of the Average Precision (AP) values obtained for all the evaluation queries. The AP of a single query *q* is calculated as follows:

$$AP(q) = \sum_{n=1}^{|G(q)|} \frac{Precision@k(q)}{|G(q)|} \quad (6)$$

In the above formula, *Precision@k(q)* is the retrieval precision over the top-*k* reports in the ranked list, i.e., the ratio of ground truth duplicate reports of the query report *q* in the top-*k* recommendation:

$$Precision@k(q) = \frac{\# \text{ ground truth in top } k}{n} \quad (7)$$

MRR (Mean Reciprocal Rank) is defined as the mean of the Reciprocal Rank (RR) values obtained for all the evaluation queries. RR of a single query *q* is the multiplicative inverse of the rank of first correct recommendation *first_q* (i.e., first ground truth duplicate report in the recommendation list):

$$RR(q) = \frac{1}{first_q} \quad (8)$$

In addition, for each evaluation metric, we obtain the *Improvement* of SETU compared with other approaches (e.g., the baseline). Taken metric *MRR* and approach *DBTM* as an example, *Improvement* is calculated as follows:

$$Improvement = \frac{MRR \text{ of SETU} - MRR \text{ of DBTM}}{MRR \text{ of DBTM}} \quad (9)$$

To further demonstrate the superiority of our proposed approach, we perform the MannWhitney U test between our proposed SETU and other approaches (i.e., the baseline). We obtain the **p-value** to demonstrate the significance of the test, and the **Cliff's delta** to demonstrate the effect size of the test. MannWhitney U test is often employed to check whether the difference in two data groups is statistically significant (which corresponds to a p-value of less than 0.05) or not. We use one-tailed MannWhitney U test with the following hypotheses:

H0: Performance produced by other approach (e.g., the baseline) is no smaller than the performance produced by SETU.

H1: Performance produced by other approach (e.g., the baseline) is smaller than the performance produced by SETU.

We include the Bonferroni correction to counteract the impact of multiple hypothesis tests.

Cliff's delta is often used to check if the difference in two data groups are substantial. Review 3-f We include the Bonferroni correction to counteract the impact of multiple hypothesis tests. The range of Cliff's delta is [-1, 1], where -1 or 1 means all values in one group are smaller or larger than those of the other group, and 0 means the data in the two groups is similar. The mappings between Cliff's delta and effectiveness levels are shown below.

Table 4: Mappings of Cliff's delta to their interpretations [9]

Cliff's delta	Interpretation
-1 ≤ Cliff's delta < 0.147	Negligible
0.147 ≤ Cliff's delta < 0.33	Small
0.33 ≤ Cliff's delta < 0.474	Medium
0.474 ≤ Cliff's delta < 1	Large

Note that, we use all results from all query reports to compute p-value and Cliff's delta. For each query report, we have one value for other approach (i.e., the baseline) and another value for our proposed approach SETU. By computing the p-value and Cliff's delta, the extent of which our approach improves over other method can be more rigorously assessed.

5.6. Parameter Setting

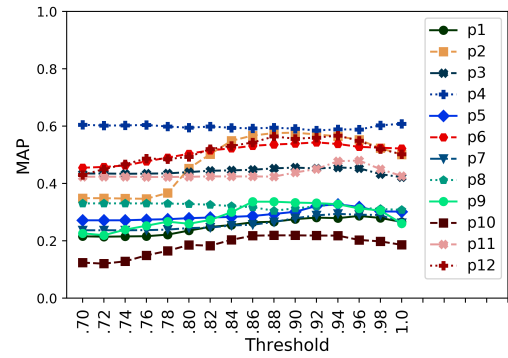


Figure 7: Influence of parameter *thresh* on duplicate detection performance

SETU has a parameter, i.e., *thresh*, to determine the reports of first class (Section 4.3). Figure 7 presents how the duplicate detection performance is influenced by different parameter values. Note that, we only use *MAP*, which is obtained considering the whole recommendation list of duplicates, to investigate the parameter's influence. Another note is that, we have experimented with *thresh* from 0.1 to 1.0, and due to space limit, we only present the results with better performance. We can easily observe that, almost for all our experimental projects, with the increase of *thresh*, the duplicate detection performance would first increase, reach a peak, and then decrease.

In our evaluation, we tune the optimal parameter value based on the training dataset (see Section 5.4) and apply it in the testing dataset to evaluate the performance of duplicate detection. In detail, for each parameter value, we first compute the average *MAP* across all the crowdtesting projects in the training dataset and treat the parameter value under which the largest average *MAP* is achieved as the optimal *thresh*. In this way, the tuned optimal parameter value is 0.94 for projects P1 - P10, and 0.92 for projects P11 and P12. For other experiments in our evaluation (i.e., *noClr*), we use the same method to tune the optimal parameter value.

6. Results and Analysis

This section presents the results and analysis of the evaluation.

6.1. Answering RQ1: Effectiveness

Table 5 presents the *recall@1*, *recall@5*, *recall@10*, *MAP*, and *MRR* for each experimental project for SETU and three baselines. To facilitate reading, Figure 8a presents the box plot of these results.

We can see that, our approach SETU can achieve the highest performance in all experimental projects for all five evaluation metrics. *recall@1* is 0.44 to 0.79 across all experimental projects, denoting in 44% to 79% circumstances, our approach can find the duplicate report in the first recommendation. *recall@5* is 0.66 to 0.92 across all experimental projects, denoting in 66% to 92% circumstances, our first five recommendations contain the duplicate report. *MAP* is 0.21 to 0.58 across all experimental projects. The reported *recall@1* in existing duplicate bug report detection approach is 0.16 to 0.67, and the reported *MAP* is 0.26 to 0.53 (see Table 2). Since our experiment is conducted on crowdtesting reports which is different from bug reports in open source projects, these figures are not comparable. However, the fact that these figures being the same order of magnitude proves the effectiveness of our approach.

Compared with the three baselines, SETU brings great improvement in all five evaluation metrics for all experimental projects. The improvement of *recall@1* is 20% to 211% compared with the three baselines, while the improvement of *MAP* is 28% to 241% in all experimental projects. Other evaluation metrics also undergo similar improvements.

We further conduct Mann-Whitney U Test for the five metrics between SETU and each baseline. compared with the three baselines, SETU statistically significantly (i.e., p-value for all tests is less than 0.05) and substantially (i.e., Cliff’s delta for all tests is large) achieves a better performance in terms of all the evaluation metrics for all experimental projects. This further indicates the effectiveness and advantage of our approach. To make our presentation more catchy, we do not provide the detailed results.

Among the three baselines, the *IR-EM* and *NextBug* employ the *product* field or *component* field to help detect duplicate reports. We use a different field *test task id* in this experiment. The reason is that the crowdtesting reports do not have the field *product* or *component*, and *test task id* is the most similar field. Moreover, we also experiment with four other fields, i.e., *phone type*, *operation system*, *ROM information*, and *network environment* (as shown in Table 1) and the performance is even worse.

The performance of *NextBug* is almost the worst. This might because the reports from different test tasks are already distinguishable in their textual descriptions. Therefore, the utilization of the *test task id* field could not provide extra information in detecting duplicate reports. The low performance of *IR-EM* in our crowdtesting reports dataset might due to the similar reason. As the *test task id* almost could not contribute to duplicate detection, the baseline *IR-EM* degenerates to, to some extent, the approach of only using textual descriptions (i.e., the TF-IDF and word embedding features, see results in Section 6.2).

The *DBTM* utilizes term-based and topic-based features for duplicate detection. The low performance of this baseline might because, unlike the large-scale open source projects, the reports

of one crowdtesting project only have very few topics. The optimal topic number is about 100 to 300 for Eclipse, OpenOffice, and Mozilla [24]. However, the optimal topic number for our experimental projects is about 5 to 10. The tiny number of topics cannot effectively help distinguish duplicate reports.

Compared with three state-of-the-art and typical baselines, SETU significantly and substantially achieves a better performance in terms of all the evaluation metrics for all experimental projects. *recall@1* is 0.44 to 0.79, *recall@5* is 0.66 to 0.92, and *MAP* is 0.21 to 0.58 across all experimental projects.

6.2. Answering RQ2: Necessity

Figure 8b and Table 6 presents the *recall@1*, *recall@5*, *recall@10*, *MAP*, and *MRR* for each experimental project for SETU and *onlyText*, *onlyImage* (see Section 5.4 for detail).

We can observe that the performance obtained by *onlyText* or *onlyImage* is worse than SETU. The improvement in *recall@1* of SETU is 23% to 211% compared with *onlyText*, and 17% to 319% compared with *onlyImage*. The improvement in *MAP* of SETU is 31% to 241% compared with *onlyText*, and 40% to 552% compared with *onlyImage*.

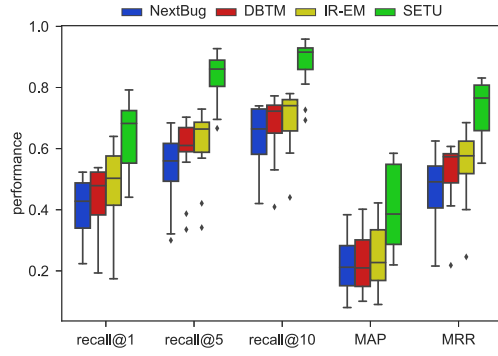
We further conduct Mann-Whitney U Test for the five metrics between SETU and *onlyText*, *onlyImage*. Compared with only using screenshots or textual descriptions (i.e., *onlyText*, *onlyImage*), SETU significantly (i.e., p-value for all tests is less than 0.05) and substantially (i.e., Cliff’s delta for all tests is large) achieves a better performance in terms of all the evaluation metrics for all experimental projects. This further indicates that only using screenshots or textual descriptions is not effective enough, and combining these two sources of information is a sensible choice for duplicate crowdtesting report detection. To make our presentation more catchy, we do not provide the detailed results either.

The performance of *onlyImage* is a little lower than the performance of *onlyText*. This might because the screenshot mainly provides the context-related information. Without the assistance of textual descriptions, the screenshot can not distinguish the duplicate reports in many circumstances. Moreover, duplicate detection with only textual descriptions can neither achieve equivalent performance with SETU, denoting the screenshot plays an indispensable role in detecting duplicate crowdtesting reports.

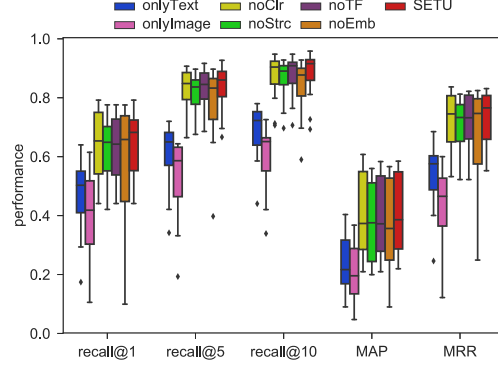
Duplicate detection with only screenshots or textual descriptions achieves significantly and substantially worse performance than SETU in all the evaluation metrics for all experimental projects. This indicates the necessity of using both screenshots and textual descriptions in detecting duplicate crowdtesting reports.

6.3. Answering RQ3: Replaceability

Figure 8b and Table 7 presents the *recall@1*, *recall@5*, *recall@10*, *MAP*, and *MRR* for each experimental project for



(a) SETU vs. baselines (RQ1)



(b) SETU vs. its alternatives (RQ2, RQ3)

Figure 8: **Review 1-e** Performance comparison

Table 5: Performance comparison between SETU and baselines (RQ1)

	SETU	IR-EM	Next Bug	DB TM	Improvement	SETU	IR-EM	Next Bug	DB TM	Improvement	SETU	IR-EM	Next Bug	DB TM	Improvement
	P1					P2					P3				
recall@1	0.792	0.640	0.440	0.424	23% - 86%	0.649	0.520	0.470	0.537	20% - 38%	0.715	0.575	0.487	0.535	24% - 46%
recall@5	0.872	0.720	0.640	0.700	21% - 36%	0.836	0.594	0.544	0.601	39% - 53%	0.894	0.654	0.504	0.602	36% - 77%
recall@10	0.944	0.780	0.728	0.750	21% - 29%	0.875	0.664	0.604	0.665	31% - 44%	0.915	0.735	0.664	0.720	24% - 37%
MAP	0.280	0.188	0.158	0.151	48% - 85%	0.570	0.336	0.283	0.389	46% - 101%	0.452	0.333	0.213	0.321	35% - 112%
MRR	0.831	0.684	0.624	0.576	21% - 44%	0.736	0.555	0.505	0.575	28% - 45%	0.794	0.616	0.576	0.601	28% - 37%
	P4					P5					P6				
recall@1	0.729	0.433	0.363	0.445	63% - 100%	0.553	0.418	0.298	0.386	32% - 85%	0.722	0.573	0.505	0.522	26% - 42%
recall@5	0.927	0.660	0.460	0.689	34% - 101%	0.815	0.675	0.575	0.615	20% - 41%	0.871	0.669	0.609	0.632	30% - 43%
recall@10	0.958	0.761	0.561	0.772	24% - 70%	0.922	0.745	0.685	0.739	23% - 34%	0.915	0.759	0.739	0.735	20% - 24%
MAP	0.584	0.171	0.271	0.198	115% - 241%	0.288	0.161	0.119	0.141	78% - 142%	0.543	0.422	0.362	0.294	28% - 84%
MRR	0.815	0.527	0.417	0.536	52% - 95%	0.663	0.490	0.320	0.447	35% - 107%	0.792	0.631	0.531	0.607	25% - 49%
	P7					P8					P9				
recall@1	0.542	0.174	0.224	0.193	141% - 211%	0.647	0.485	0.415	0.521	24% - 55%	0.549	0.403	0.353	0.372	36% - 55%
recall@5	0.666	0.341	0.300	0.335	95% - 122%	0.849	0.667	0.583	0.605	27% - 45%	0.768	0.568	0.528	0.555	35% - 45%
recall@10	0.693	0.440	0.420	0.409	57% - 69%	0.877	0.702	0.664	0.712	23% - 32%	0.811	0.637	0.587	0.606	27% - 38%
MAP	0.259	0.090	0.080	0.100	159% - 223%	0.321	0.244	0.164	0.220	31% - 95%	0.307	0.210	0.210	0.195	46% - 57%
MRR	0.565	0.245	0.215	0.218	130% - 162%	0.738	0.597	0.473	0.571	23% - 56%	0.644	0.527	0.477	0.501	22% - 35%
	P10					P11					P12				
recall@1	0.440	0.293	0.223	0.257	50% - 97%	0.773	0.613	0.523	0.512	26% - 50%	0.719	0.578	0.488	0.525	24% - 47%
recall@5	0.695	0.421	0.321	0.387	65% - 116%	0.887	0.729	0.659	0.662	21% - 34%	0.927	0.724	0.684	0.702	28% - 35%
recall@10	0.726	0.585	0.486	0.530	24% - 49%	0.924	0.759	0.739	0.725	21% - 27%	0.948	0.764	0.734	0.750	24% - 29%
MAP	0.219	0.159	0.129	0.143	37% - 69%	0.450	0.332	0.282	0.274	35% - 64%	0.564	0.413	0.383	0.401	36% - 47%
MRR	0.552	0.400	0.370	0.412	33% - 49%	0.828	0.621	0.531	0.577	33% - 55%	0.805	0.646	0.586	0.601	24% - 37%

Table 6: Performance comparison between SETU and *onlyText*, *onlyImage* (RQ2)

	SETU	only Text	only Im- age	Improvement	SETU	only Text	only Im- age	Improvement	SETU	only Text	only Im- age	Improvement	SETU	only Text	only Im- age	Improvement
	P1				P2				P3				P4			
recall@1	0.792	0.640	0.560	23% - 41%	0.649	0.520	0.505	24% - 28%	0.715	0.550	0.457	29% - 56%	0.729	0.433	0.397	68% - 83%
recall@5	0.872	0.720	0.640	21% - 36%	0.836	0.584	0.550	43% - 51%	0.894	0.634	0.568	41% - 57%	0.927	0.660	0.643	40% - 44%
recall@10	0.944	0.780	0.664	21% - 42%	0.875	0.646	0.601	35% - 45%	0.915	0.715	0.647	27% - 41%	0.958	0.761	0.725	25% - 32%
MAP	0.280	0.188	0.165	48% - 69%	0.570	0.326	0.257	74% - 121%	0.452	0.313	0.283	44% - 59%	0.584	0.171	0.178	228% - 241%
MRR	0.831	0.684	0.600	21% - 38%	0.736	0.555	0.422	32% - 74%	0.794	0.596	0.512	33% - 55%	0.815	0.527	0.467	54% - 74%
	P5				P6				P7				P8			
recall@1	0.553	0.418	0.363	32% - 52%	0.722	0.553	0.614	17% - 30%	0.542	0.174	0.184	194% - 211%	0.647	0.485	0.342	33% - 89%
recall@5	0.815	0.675	0.630	20% - 29%	0.871	0.639	0.635	36% - 37%	0.666	0.341	0.331	95% - 101%	0.849	0.667	0.488	27% - 73%
recall@10	0.922	0.745	0.660	23% - 39%	0.915	0.749	0.689	22% - 32%	0.693	0.440	0.420	57% - 64%	0.877	0.702	0.511	24% - 71%
MAP	0.288	0.161	0.147	78% - 95%	0.543	0.402	0.367	35% - 47%	0.259	0.090	0.080	187% - 223%	0.321	0.244	0.212	31% - 51%
MRR	0.663	0.490	0.463	35% - 43%	0.792	0.611	0.569	29% - 39%	0.565	0.245	0.235	130% - 140%	0.738	0.597	0.407	23% - 81%
	P9				P10				P11				P12			
recall@1	0.549	0.383	0.138	43% - 297%	0.440	0.293	0.105	50% - 319%	0.773	0.593	0.555	30% - 39%	0.719	0.538	0.438	33% - 64%
recall@5	0.768	0.528	0.193	45% - 297%	0.695	0.421	0.391	65% - 77%	0.887	0.709	0.622	25% - 42%	0.927	0.704	0.604	31% - 53%
recall@10	0.811	0.617	0.339	31% - 139%	0.726	0.585	0.565	24% - 28%	0.924	0.729	0.655	26% - 41%	0.948	0.764	0.664	24% - 42%
MAP	0.307	0.180	0.047	70% - 553%	0.219	0.159	0.096	37% - 128%	0.450	0.312	0.321	40% - 44%	0.564	0.403	0.303	39% - 86%
MRR	0.644	0.477	0.121	35% - 432%	0.552	0.400	0.229	38% - 141%	0.828	0.601	0.593	37% - 39%	0.805	0.606	0.506	32% - 59%

SETU and *noClr*, *noStrc*, *noTF*, *noEmb* (see Section 5.4 for detail).

We can observe that, in most circumstances, performance obtained by SETU is better than or equal with the performance

Table 7: Performance comparison between SETU and *noClr*, *noSrc*, *noTF*, *noEmb* (RQ3)

	SETU	noClr	noSrc	noTF	noEmb	Imprv.	SETU	noClr	noSrc	noTF	noEmb	Imprv.	SETU	noClr	noSrc	noTF	noEmb	Imprv.
	P1						P2						P3					
recall@1	0.792	0.792	0.712	0.776	0.768	0% - 11%	0.649	0.649	0.634	0.648	0.653	0% - 2%	0.715	0.657	0.663	0.626	0.663	7% - 14%
recall@5	0.872	0.870	0.840	0.860	0.860	0% - 3%	0.836	0.820	0.817	0.830	0.807	0% - 3%	0.894	0.884	0.857	0.884	0.857	1% - 4%
recall@10	0.944	0.928	0.928	0.940	0.928	0% - 1%	0.875	0.873	0.850	0.869	0.860	0% - 2%	0.915	0.905	0.889	0.901	0.905	1% - 2%
MAP	0.280	0.277	0.232	0.269	0.268	1% - 20%	0.570	0.573	0.537	0.563	0.555	-0% - 6%	0.452	0.434	0.435	0.432	0.435	3% - 4%
MRR	0.831	0.836	0.781	0.821	0.816	-0% - 6%	0.736	0.734	0.718	0.726	0.735	0% - 2%	0.794	0.755	0.747	0.737	0.757	4% - 7%
	P4						P5						P6					
recall@1	0.729	0.760	0.739	0.729	0.750	-4% - 0%	0.553	0.541	0.482	0.529	0.458	2% - 20%	0.722	0.746	0.674	0.726	0.734	-3% - 7%
recall@5	0.927	0.906	0.875	0.916	0.864	1% - 7%	0.815	0.809	0.767	0.813	0.750	0% - 8%	0.871	0.867	0.839	0.859	0.867	0% - 3%
recall@10	0.958	0.947	0.927	0.947	0.906	1% - 5%	0.922	0.904	0.898	0.916	0.821	0% - 12%	0.915	0.903	0.891	0.915	0.895	0% - 2%
MAP	0.584	0.607	0.559	0.583	0.566	-3% - 4%	0.288	0.287	0.247	0.281	0.253	0% - 16%	0.547	0.547	0.502	0.531	0.534	-0% - 8%
MRR	0.815	0.827	0.806	0.819	0.809	-1% - 1%	0.663	0.655	0.608	0.669	0.589	-0% - 12%	0.792	0.801	0.754	0.795	0.792	-1% - 5%
	P7						P8						P9					
recall@1	0.542	0.440	0.420	0.440	0.400	23% - 35%	0.647	0.612	0.603	0.636	0.578	1% - 11%	0.549	0.540	0.575	0.540	0.416	-4% - 31%
recall@5	0.666	0.665	0.675	0.685	0.655	-2% - 1%	0.849	0.829	0.833	0.826	0.780	1% - 8%	0.768	0.746	0.781	0.742	0.648	-1% - 18%
recall@10	0.693	0.706	0.696	0.706	0.696	-1% - 0%	0.877	0.862	0.864	0.867	0.841	1% - 4%	0.811	0.798	0.824	0.793	0.759	-1% - 6%
MAP	0.259	0.209	0.199	0.209	0.179	23% - 44%	0.321	0.311	0.283	0.312	0.277	2% - 15%	0.307	0.304	0.315	0.295	0.236	-2% - 30%
MRR	0.565	0.532	0.522	0.522	0.502	6% - 12%	0.738	0.718	0.703	0.726	0.672	1% - 9%	0.644	0.633	0.666	0.633	0.530	-3% - 21%
	P10						P11						P12					
recall@1	0.440	0.443	0.480	0.484	0.099	-9% - 344%	0.773	0.765	0.775	0.761	0.775	-0% - 1%	0.719	0.719	0.699	0.699	0.709	0% - 2%
recall@5	0.695	0.664	0.714	0.732	0.397	-5% - 75%	0.887	0.889	0.868	0.883	0.879	-0% - 2%	0.927	0.907	0.897	0.907	0.897	2% - 3%
recall@10	0.726	0.712	0.747	0.763	0.590	-4% - 23%	0.924	0.922	0.910	0.926	0.893	-0% - 3%	0.948	0.928	0.908	0.918	0.898	2% - 5%
MAP	0.219	0.211	0.226	0.242	0.089	-9% - 146%	0.450	0.436	0.444	0.447	0.442	0% - 3%	0.564	0.554	0.544	0.544	0.524	1% - 7%
MRR	0.552	0.554	0.573	0.588	0.249	-6% - 121%	0.828	0.823	0.812	0.821	0.824	0% - 1%	0.805	0.785	0.775	0.805	0.785	0% - 3%

Table 8: Results of Mann-Whitney U Test between SETU and *noClr*, *noSrc*, *noTF*, *noEmb* (RQ3)

	SETU vs. noClr	SETU vs. noSrc	SETU vs. noTF	SETU vs. noEmb	SETU vs. noClr	SETU vs. noSrc	SETU vs. noTF	SETU vs. noEmb	SETU vs. noClr	SETU vs. noSrc	SETU vs. noTF	SETU vs. noEmb	SETU vs. noClr	SETU vs. noSrc	SETU vs. noTF	SETU vs. noEmb
	P1				P2				P3				P4			
recall@1	0.05 (0.11 N)	0.00 (0.41 M)	0.05 (0.11 N)	0.00 (0.30 S)	0.71 (-0.0 N)	0.31 (0.02 N)	0.48 (0.00 N)	0.67 (-0.0 N)	0.00 (0.24 S)	0.00 (0.29 S)	0.00 (0.42 M)	0.00 (0.28 S)	0.00 (0.19 S)	0.00 (0.18 S)	0.36 (0.02 N)	0.02 (0.14 N)
recall@5	0.88 (-0.0 N)	0.00 (0.18 S)	0.36 (0.02 N)	0.02 (0.14 N)	0.02 (0.11 N)	0.36 (0.01 N)	0.50 (-0.0 N)	0.01 (0.12 N)	0.05 (0.09 N)	0.00 (0.19 S)	0.12 (0.06 N)	0.00 (0.21 S)	0.00 (0.17 S)	0.00 (0.17 S)	0.09 (0.07 N)	0.09 (0.07 N)
recall@10	0.37 (0.02 N)	0.82 (-0.0 N)	0.71 (-0.0 N)	0.52 (-0.0 N)	0.91 (-0.0 N)	0.04 (0.09 N)	0.71 (-0.0 N)	0.28 (0.03 N)	0.09 (0.07 N)	0.00 (0.17 S)	0.00 (0.14 N)	0.09 (0.07 N)	0.01 (0.15 S)	0.00 (0.48 L)	0.02 (0.14 N)	0.00 (0.27 S)
MAP	0.01 (0.15 S)	0.00 (0.48 L)	0.02 (0.14 N)	0.00 (0.27 S)	0.66 (-0.0 N)	0.00 (0.29 S)	0.12 (0.06 N)	0.00 (0.13 N)	0.01 (0.12 N)	0.00 (0.18 S)	0.06 (0.09 N)	0.00 (0.15 S)	0.79 (-0.0 N)	0.00 (0.24 S)	0.24 (0.05 N)	0.67 (-0.0 N)
MRR	0.79 (-0.0 N)	0.00 (0.24 S)	0.24 (0.05 N)	0.67 (-0.0 N)	0.25 (0.03 N)	0.01 (0.12 N)	0.24 (0.03 N)	0.25 (0.03 N)	0.00 (0.19 S)	0.00 (0.28 S)	0.00 (0.26 S)	0.00 (0.32 S)				
	P5				P6				P7				P8			
recall@1	0.99 (-0.2 N)	0.89 (-0.1 N)	0.49 (0.00 N)	0.99 (-0.3 N)	0.16 (0.06 N)	0.00 (0.41 M)	0.00 (0.20 S)	0.00 (0.53 L)	0.61 (-0.0 N)	0.00 (0.37 M)	0.31 (0.02 N)	0.87 (-0.0 N)	0.16 (0.08 N)	0.00 (0.23 S)	0.56 (-0.0 N)	0.02 (0.16 S)
recall@5	0.16 (0.08 N)	0.00 (0.23 S)	0.56 (-0.0 N)	0.02 (0.16 S)	0.09 (0.08 N)	0.00 (0.27 S)	0.04 (0.10 N)	0.00 (0.36 M)	0.06 (0.07 N)	0.00 (0.18 S)	0.04 (0.09 N)	0.96 (-0.0 N)	0.05 (0.12 N)	0.00 (0.23 S)	0.00 (0.20 S)	0.00 (0.34 M)
recall@10	0.05 (0.12 N)	0.00 (0.23 S)	0.00 (0.20 S)	0.00 (0.34 M)	0.09 (0.08 N)	0.01 (0.14 N)	0.29 (0.03 N)	0.00 (0.48 L)	0.14 (0.05 N)	0.01 (0.10 N)	0.84 (-0.0 N)	0.11 (0.06 N)	0.99 (-0.2 N)	0.02 (0.17 S)	0.56 (-0.0 N)	0.40 (0.02 N)
MAP	0.99 (-0.2 N)	0.00 (0.17 S)	0.56 (-0.0 N)	0.40 (0.02 N)	0.38 (0.01 N)	0.00 (0.37 M)	0.12 (0.07 N)	0.00 (0.36 M)	0.53 (-0.0 N)	0.00 (0.16 S)	0.39 (0.01 N)	0.44 (0.00 N)	0.60 (-0.0 N)	0.41 (0.01 N)	0.90 (-0.1 N)	0.18 (0.07 N)
MRR	0.60 (-0.0 N)	0.00 (0.24 S)	0.24 (0.05 N)	0.67 (-0.0 N)	0.40 (0.01 N)	0.00 (0.19 S)	0.99 (-0.1 N)	0.00 (0.37 M)	0.55 (-0.0 N)	0.00 (0.27 S)	0.52 (-0.0 N)	0.26 (0.03 N)				
	P9				P10				P11				P12			
recall@1	0.00 (0.57 L)	0.00 (0.69 L)	0.00 (0.61 L)	0.00 (0.83 L)	0.00 (0.19 S)	0.00 (0.23 S)	0.06 (0.06 N)	0.00 (0.30 S)	0.01 (0.06 N)	0.99 (-0.1 N)	0.00 (0.07 N)	0.00 (0.72 L)	0.00 (0.19 S)	0.57 (-0.0 N)	0.77 (-0.0 N)	0.48 (0.00 N)
recall@5	0.19 (0.10 N)	0.00 (0.23 S)	0.77 (-0.0 N)	0.48 (0.00 N)	0.09 (0.05 N)	0.09 (0.05 N)	0.00 (0.13 N)	0.00 (0.32 S)	0.00 (0.07 N)	0.95 (-0.0 N)	0.00 (0.13 N)	0.00 (0.56 L)	0.33 (0.05 N)	0.55 (-0.0 N)	0.53 (-0.0 N)	0.29 (0.06 N)
recall@10	0.33 (0.05 N)	0.55 (-0.0 N)	0.53 (-0.0 N)	0.29 (0.06 N)	0.24 (0.02 N)	0.06 (0.06 N)	0.17 (0.03 N)	0.00 (0.16 S)	0.00 (0.08 N)	0.79 (-0.0 N)	0.00 (0.09 N)	0.00 (0.32 S)	MAP	0.00 (0.53 L)	0.00 (0.64 L)	0.00 (0.47 M)
MAP	0.00 (0.53 L)	0.00 (0.64 L)	0.00 (0.47 M)	0.00 (0.69 L)	0.01 (0.08 N)	0.00 (0.32 S)	0.00 (0.11 N)	0.00 (0.40 M)	0.00 (0.08 N)	0.96 (-0.0 N)	0.00 (0.17 S)	0.00 (0.63 L)	MRR	0.05 (0.18 S)	0.00 (0.32 S)	0.00 (0.30 S)
MRR	0.05 (0.18 S)	0.00 (0.32 S)	0.00 (0.30 S)	0.00 (0.34 M)	0.00 (0.11 N)	0.00 (0.18 S)	0.29 (0.03 N)	0.37 (0.01 N)	0.03 (0.06 N)	0.99 (-0.2 N)	0.42 (0.00 N)	0.00 (0.64 L)				
	P13				P14				P15				P16			
recall@1	0.78 (-0.0 N)	0.99 (-0.3 N)	0.99 (-0.3 N)	0.00 (1.0 L)	0.09 (0.08 N)	0.79 (-0.0 N)	0.17 (0.06 N)	0.66 (-0.0 N)	0.13 (0.05 N)	0.00 (0.13 N)	0.00 (0.12 N)	0.02 (0.10 N)	recall@5	0.00 (0.19 S)	0.99 (-0.1 N)	0.99 (-0.1 N)
recall@5	0.00 (0.19 S)	0.99 (-0.1 N)	0.99 (-0.1 N)	0.00 (0.98 L)	0.45 (0.00 N)	0.10 (0.07 N)	0.20 (0.05 N)	0.15 (0.06 N)	0.00 (0.16 S)	0.00 (0.20 S)	0.00 (0.12 N)	0.00 (0.21 S)	recall@10	0.11 (0.06 N)	0.98 (-0.1 N)	0.99 (-0.1 N)
recall@10	0.11 (0.06 N)	0.98 (-0.1 N)	0.99 (-0.1 N)	0.00 (0.57 L)	0.53 (-0.0 N)	0.57 (-0.0 N)	0.80 (-0.0 N)	0.06 (0.09 N)	0.02 (0.10 N)	0.00 (0.20 S)	0.00 (0.20 S)	0.00 (0.22 S)	MAP	0.15 (0.05 N)	0.96 (-0.0 N)	0.98 (-0.1 N)
MAP	0.15 (0.05 N)	0.96 (-0.0 N)	0.98 (-0.1 N)	0.00 (0.90 L)	0.39 (0.01 N)	0.58 (-0.0 N)	0.31 (0.03 N)	0.56 (-0.0 N)	0.27 (0.03 N)	0.00 (0.15 S)	0.00 (0.13 N)	0.00 (0.25 S)	MRR	0.53 (-0.0 N)	0.97 (-0.1 N)	0.99 (-0.2 N)
MRR	0.53 (-0.0 N)	0.97 (-0.1 N)	0.99 (-0.2 N)	0.00 (0.99 L)	0.97 (-0.1 N)	0.17 (0.06 N)	0.29 (0.03 N)	0.37 (0.01 N)	0.00 (0.13 N)	0.00 (0.16 S)	0.49 (0.00 N)	0.00 (0.13 N)				

Note that: The figures X (Y Z) respectively denotes p-value, Cliffs delta, and interpretation of Cliffs delta (i.e., Large (L), Medium (M), Small (S), and Negligible (N))

obtained by using three features (i.e., *noClr*, *noSrc*, *noTF*, or *noEmb*). In rare cases, the performance obtained by using three features is a little better than the performance of SETU.

We further conduct Mann-Whitney U Test for the five metrics between SETU and *noClr*, *noSrc*, *noTF*, *noEmb*. Table 8 shows the p-value, the Cliffs delta, and the interpretation of these tests (see Section 5.5 for details). There are totally 240 tests (12 projects, 4 types of features, 5 evaluation metrics). Among them, in 35% (83/240) tests, the performance obtained by SETU is significantly (i.e., p-value is less than 0.05) and substantially (i.e., Cliffs delta is not negligible, i.e., small in 21% tests, median in 5% tests, or large in 9% tests) better than the performance of using three features. In other 65% (157/240) tests, the performance obtained by SETU demonstrates negligible difference (i.e., Cliff's delta is negligible although some p-value is less than 0.05) with the performance of using three features. In none of the 240 tests, the performance of using three features is significantly and substantially better than the performance of SETU. This further indicates our proposed ap-

proach of combining these four types of features is effective.

Among the four experiments with three types of features, we can see that *noEmb* achieves relatively worse results. This might because the word embedding feature focuses on the relationship of terms by considering the context they appear. Without this feature (i.e., *noEmb*), simply matching the occurrence of terms, which is done by TF-IDF feature, cannot effectively detecting the duplicate reports. This implies word embedding feature is the least replaceable feature, i.e., the performance would undergo a relatively large decline if removing this feature.

We can also see that *noClr* achieves relatively better results among the four experiments with three types of features. This indicates image color feature is the most replaceable feature, i.e., the performance would undergo a relatively small decrease if removing this feature. This also implies that image color feature would sometimes bring noise in the duplicate detection. We will present further discussion in Section 7.3.

Word embedding is the least replaceable feature, while image color is the most replaceable feature. In addition, duplicate detection with all the four features can achieve relatively best performance.

7. Discussion

7.1. Further Exploration of Results

Review 1-f

We have noticed that the performance of duplicate detection differ in a large range for the projects. We conducted detailed analysis and summarized two typical reasons for the low performance in some projects.

Firstly, the low quality of screenshots could not effectively provide context-related information. We find that in some projects (e.g., P7, P9), the screenshots of many crowdtesting reports do not match with the textual descriptions. For example, a report describes the short message firewall in its textual description. But its screenshot is about the message sending page (i.e., an intermediate step for the textual description). SETU can be misled by these inaccurate screenshots. This is mainly because the crowdworkers are unprofessional in software testing, and proper training and guidance are needed in crowdtesting [43].

Secondly, the short textual descriptions can only provide limited information and make it hard for duplicate detection. Analytical tasks (such as classification, clustering, etc.) involving short texts have long been recognized as challenging due to the lack of context and owing to their sparseness [2, 32]. We find that in project P1, the median term length of crowdtesting reports is 16, and the median term length of reports of project P10 is only 9. This is another important reason for the low duplicate detection performance of P10.

However note that, existing duplicate detection approaches also suffer from the performance variation in different experimental projects. For example, the recall@1 is 42% - 57% in [24], and the MAP is 29% - 51% in [17].

7.2. Further Exploration of Image Features

We have mentioned that *noClr* (i.e., duplicate detection without image color feature) can sometimes achieve a slightly better performance than SETU, e.g., in P4, P6 (see details in Table 7). This indicates that image color feature would bring noise to the duplicate detection in some cases.

We have examined the screenshots of the experimental projects, and found that, in these projects (i.e., P4, P6), screenshots of different functionalities (i.e., denoting non-duplicate reports) can sometimes have very similar color distribution (as Figure 6b shows). This is somehow coincident with our common sense, because many apps pursue a unified interface design to make it more user-friendly. Under this case, the non-duplicate reports might share very similar image color features, and removing this feature (i.e., *noClr*) can instead increase the detection performance.

More than that, we also noticed that, in some projects as P9, P10, the performance achieved by *noStrc* (i.e., duplicate detection without image structure feature) is a little better than SETU.

We further examined the screenshots of these two projects (i.e., P9, P10), and found that, the images' structure for different functionalities (i.e., denoting non-duplicate reports) can sometimes exert no much difference, just like Figure 5b shows. In this case, the non-duplicate reports would share very similar image structure features, and removing the structure feature (i.e., *noStrc*) can instead increase the detection performance.

The above analysis indicates that image color feature can occasionally bring noise and removing it can increase the performance for duplicate detection, and so does the image structure feature. We have also analyzed the reasons and found that if non-duplicate reports tend to share a similar color distribution, removing image color feature can achieve a better performance; this also holds true for image structure feature when non-duplicate reports usually share a similar image structure.

This motivates us to conduct feature selection for specific crowdtesting projects to further improve the duplicate detection performance. In practice, this can be done based on the received reports of the project during the crowdtesting process. One can examine whether a large number of non-duplicate reports share the similar color distribution or structure, so as to suggest which features to be used for detecting the new-coming reports for this specific project.

7.3. Advantage of Hierarchical Approach

Our proposed SETU employs a hierarchical algorithm to detect duplicate reports, i.e., first classify the reports into two classes, and rank them separately. Under SETU, the class information indicates different levels of certainty of being duplicates. In detail, the first class contains the reports whose screenshot similarity is large enough, thus they can be more likely to be the duplicates of the query report. On the contrary, the second class contains the reports whose screenshots are quite different with the query report, thus they are less likely to be the duplicates of the query report.

Therefore, when we provide the users with the duplicate detection results, these class information can give more insights of the detection. For example, we encourage the users put more focus on the first class since they have large probability to be the true duplicates. For the second class, the users only need to glance off the reports to examine whether they are the true duplicates, especially when there have been a large number of recommended reports in the first class. We believe our two-classes hierarchical approach can provide more practical assistance for duplicate detection in real-world crowdtesting practice. Meanwhile, existing approaches treated all the reports as a whole, thus could not provide this kinds of guidelines.

7.4. Alternative Combination Manner

The motivating examples in Section 2.2 indicate that the role of screenshots is mainly to demonstrate the context-related information, while the role of textual descriptions is to provide

detailed illustration of the reported problem. This is why our proposed approach SETU first uses screenshot similarity to filter the reports to first class and conducts the ranking separately. However, one may still argue that other combination manners could achieve better performance. We did conduct experiments to explore whether other combination manners could outperform SETU in duplicate detection. Due to word limit, we provide the detailed results and analysis, as well as the experimental setup in the external link¹⁰. Here, we briefly summarize the main findings.

We had designed three new combination manners, i.e., *addCmb*, *multiplyCmb*, and *textFirst*. Specifically, *addCmb* denotes adding screenshot similarity and textual similarity as one similarity value and ranking the reports based on it, which is a straight-forward manner. *multiplyCmb* denotes multiplying the screenshot similarity with textual similarity as one similarity value and ranking the reports based on it, which is borrowed from [42]. *textFirst* denotes treating the reports with high textual similarity as the first class and ranking them with the screenshot similarity (the second class is treated as SETU does).

We further conduct Mann-Whitney U Test for the five metrics between SETU and *addCmb*, *multiplyCmb*, *textFirst*. There are totally 180 tests (12 projects, 3 alternative combination manners, 5 evaluation metrics). Among them, in 73% (131/180) tests, the performance obtained by SETU is significantly (i.e., p-value is less than 0.05) and substantially (i.e., Cliff's delta is not negligible, i.e., small in 30% tests, median in 15% tests, or large in 28% tests) better than the performance of other combination manners. In other 27% (49/180) tests, the performance obtained by SETU demonstrates negligible difference (i.e., Cliff's delta is negligible although some p-value is less than 0.05) with the performance of other combination manners. In none of the 180 tests, the performance of alternative combination manners is significantly and substantially better than the performance of SETU. To summarize, the combination manner proposed in SETU can achieve relatively highest performance than other alternative combination manners.

7.5. Threats to Validity

The external threats concern the generality of this study. First, our experiment data consists of 12 projects collected from one of the Chinese largest crowdtesting platforms. We can not ensure that the results of our study could generalize beyond this environment in which it was conducted. However, the various domains of projects and size of data relatively reduce this risk. Second, all crowdtesting reports investigated in this study are written in Chinese, and we cannot assure that similar results can be observed on crowdtesting projects in other languages. But this is alleviated due to the fact that we did not conduct semantic comprehension, but rather simply tokenize sentence and use word as token for experiment.

Internal validity of this study mainly questions the selection and implementation of baselines. Because there is no approach

for duplicate *crowdtesting report* detection, we can only employ the approaches for duplicate *bug report* detection. Moreover, as the crowdtesting reports do not have the *product* and *component* fields as the original approaches, we employ the most similar field (i.e., *test task id*) for substitution. In addition, as these three baseline approaches are not publicly available, we implement our own versions rigorously following the steps described in their papers.

Construct validity of this study mainly questions the data processing method. We rely on the stored duplicate labels of crowdtesting reports to construct the ground truth. However, this is addressed to some extent due to the fact that testers in the company have no knowledge that this study will be performed for them to artificially modify their labeling. Besides, we have verified its validity through random sampling and relabeling.

8. Conclusion

In this work, we propose SETU, which combines the information from both the screenshots and the textual descriptions to detect duplicate crowdtesting reports. We evaluate the effectiveness of SETU on 12 commercial projects with 3,689 reports from one of the Chinese largest crowdtesting platforms, and the results are promising.

Review 1-d

Review 2-a

Note that, the application scope of SETU is not limited to crowdtesting reports. It can also be applied in traditional software testing, especially testing of GUI system, where the screenshots can be easily obtained. The adoption of SETU in other scenarios and further demonstration of its feasibility will be conducted in future.

This paper is just the starting point of the work in progress. We are closely collaborating with Baidu CrowdTest crowdtesting platform and planning to deploy the proposed duplicate crowdtesting report detection approach online. The feedback from real-world case studies will further validate the effectiveness, as well as guide us in improving SETU. For the future work, we would like to explore other features, as well as conduct intelligent selection of optimal features to further improve the duplicate detection performance.

9. Acknowledgments

This work is supported by the National Natural Science Foundation of China under grant No.61602450, No.6143200, and China Scholarship Council. We would like to thank the testers in Baidu for their great efforts in supporting this work.

10. Reference

- [1] Alipour, A., Hindle, A., Stroulia, E., 2013. A contextual approach towards more accurate duplicate bug report detection, in: MSR'13, pp. 183–192.
- [2] Bairi, R.B., Udupa, R., Ramakrishnan, G., 2016. A framework for task-specific short document expansion, in: CIKM'2016, pp. 791–800.
- [3] Banerjee, S., Cukic, B., Adjeroh, D., 2012. Automated duplicate bug report classification using subsequence matching, in: IEEE 14th International Symposium on High-Assurance Systems Engineering, IEEE. pp. 74–81.

¹⁰<https://github.com/wangjunjieISCAS/ISTDuplicateDetection>

- [4] Banerjee, S., Syed, Z., Helmick, J., Cukic, B., 2013. A fusion approach for classifying duplicate problem reports, in: ISSRE'13, pp. 208–217.
- [5] Bengio, Y., Ducharme, R., Vincent, P., Janvin, C., 2003. A neural probabilistic language model. *The Journal of Machine Learning Research* 3, 1137–1155.
- [6] Berson, A., Smith, S., Thearling, K., 2004. An overview of data mining techniques. *Building Data Mining Application for CRM*.
- [7] Chen, N., Kim, S., 2012. Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles, in: ASE'12, pp. 140–149.
- [8] Chen, X., Lawrence Zitnick, C., 2015. Mind's eye: A recurrent visual representation for image caption generation, in: CVPR'15, pp. 2422–2431.
- [9] Cliff, N., 2014. Ordinal methods for behavioral data analysis. Psychology Press.
- [10] Cui, Q., Wang, J., Yang, G., Xie, M., Wang, Q., Li, M., 2017a. Who should be selected to perform a task in crowdsourced testing?, in: COMP-SAC'17, pp. 75–84.
- [11] Cui, Q., Wang, S., Wang, J., Hu, Y., Wang, Q., Li, M., 2017b. Multi-objective crowd worker selection in crowdsourced testing, in: SEKE'17, pp. 1–6.
- [12] Feng, Y., Chen, Z., Jones, J.A., Fang, C., Xu, B., 2015. Test report prioritization to assist crowdsourced testing, in: FSE'15, pp. 225–236.
- [13] Feng, Y., Jones, J.A., Chen, Z., Fang, C., 2016. Multi-objective test report prioritization using image understanding, in: ASE'16, pp. 202–213.
- [14] G., M., R., R., A., B., S., L., 2016. Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring, in: MOBILESoft'16, pp. 88–99.
- [15] Gao, R., Wang, Y., Feng, Y., Chen, Z., Wong, W.E., 2018. Successes, challenges, and rethinking—an industrial investigation on crowdsourced mobile application testing. *Empirical Software Engineering*, 1–25.
- [16] Gomide, V.H.M., Valle, P.A., Ferreira, J.O., Barbosa, J.R.G., da Rocha, A.F., d. A. Barbosa, T.M.G., 2014. Affective crowdsourcing applied to usability testing. *Int. J. of Computer Science and Information Technologies* 5, 575–579.
- [17] Hindle, A., Alipour, A., Stroulia, E., 2016. A contextual approach towards more accurate duplicate bug report detection and ranking. *Empirical Software Engineering* 21, 368–410.
- [18] Jalbert, N., Weimer, W., 2008. Automated duplicate detection for bug tracking systems, in: IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, pp. 52–61.
- [19] Jiang, H., Chen, X., He, T., Chen, Z., Li, X., 2018. Fuzzy clustering of crowdsourced test reports for apps. *TOIT'2018*, 18:1–28.
- [20] Liu, D., Zhang, X., Feng, Y., Jones, J.A., 2018. Generating descriptions for screenshots to assist crowdsourced testing, in: SANER'2018, pp. 492–496.
- [21] Manjunath, B.S., Ohm, J.R., Vasudevan, V.V., Yamada, A., 2001. Color and texture descriptors. *IEEE Trans. Cir. and Sys. for Video Technol.* 11, 703–715.
- [22] Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., 2013. Distributed representations of words and phrases and their compositionality, in: NIPS'13, pp. 3111–3119.
- [23] Musson, R., Richards, J., Fisher, D., Bird, C., Bussone, B., Ganguly, S., 2013. Leveraging the crowd: How 48,000 users helped improve lync performance. *IEEE Software* 30, 38–45.
- [24] Nguyen, A.T., Nguyen, T.T., Nguyen, T.N., Lo, D., Sun, C., 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling, in: ASE'12, pp. 70–79.
- [25] Oliva, A., Torralba, A., 2001. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision* 42, 145–175.
- [26] Prifti, T., Banerjee, S., Cukic, B., 2011. Detecting bug duplicate reports through local references, in: Proceedings of the 7th International Conference on Predictive Models in Software Engineering, pp. 8:1–10.
- [27] Rocha, H., Valente, M.T., Marques-Neto, H., Murphy, G.C., 2016. An empirical study on recommendations of similar bugs, in: SANER'16, pp. 46–56.
- [28] Runeson, P., Alexandersson, M., Nyholm, O., 2007. Detection of duplicate defect reports using natural language processing, in: ICSE'07, pp. 499–510.
- [29] Sun, C., Lo, D., Khoo, S., Jiang, J., 2011. Towards more accurate retrieval of duplicate bug reports, in: ASE'11, pp. 253–262.
- [30] Sun, C., Lo, D., Wang, X., Jiang, J., Khoo, S., 2010. A discriminative model approach for accurate duplicate bug report retrieval, in: ICSE'10, pp. 45–54.
- [31] Sureka, A., Jalote, P., 2010. Detecting duplicate bug report using character n-gram-based features, in: APSEC'10, pp. 366–374.
- [32] Tang, J., Wang, Y., Zheng, K., Mei, Q., 2017. End-to-end learning for short text expansion, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1105–1113.
- [33] Tian, Y., Sun, C., Lo, D., 2012. Improved duplicate bug report identification, in: CSMR'12, pp. 385–390.
- [34] Vinyals, O., Toshev, A., Bengio, S., Erhan, D., 2017. Show and tell: Lessons learned from the 2015 mscoco image captioning challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39, 652–663.
- [35] Wang, J., Cui, Q., Wang, Q., Wang, S., 2016a. Towards effectively test report classification to assist crowdsourced testing, in: ESEM'16, pp. 6:1–6:10.
- [36] Wang, J., Cui, Q., Wang, S., Wang, Q., 2017. Domain adaptation for test report classification in crowdsourced testing, in: ICSE-SEIP'17, pp. 83–92.
- [37] Wang, J., Wang, S., Cui, Q., Wang, Q., 2016b. Local-based active classification of test report to assist crowdsourced testing, in: ASE'16, pp. 190–201.
- [38] Wang, X., Zhang, L., Xie, T., Anvik, J., Sun, J., 2008. An approach to detecting duplicate bug reports using natural language and execution information, in: ICSE'08, pp. 461–470.
- [39] Witten, I.H., Frank, E., 2005. *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann.
- [40] Xie, M., Wang, Q., Yang, G., Li, M., 2017. Cocoon: Crowdsourced testing quality maximization under context coverage constraint, in: ISSRE'17, pp. 316–327.
- [41] Xu, B., Ye, D., Xing, Z., Xia, X., Chen, G., Li, S., 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network, in: ASE'16, pp. 51–62.
- [42] Yang, X., Lo, D., Xia, X., Bao, L., Sun, J., 2016. Combining word embedding with information retrieval to recommend similar bug reports, in: ISSRE'16, pp. 127–137.
- [43] Zhang, X., Chen, Z., Fang, C., Liu, Z., 2016. Guiding the crowds for android testing, in: ICSE'2016, pp. 752–753.
- [44] Zhou, J., Zhang, H., 2012. Learning to rank duplicate bug reports, in: Proceedings of the 21st ACM international conference on Information and knowledge management, pp. 852–861.