CTRAS: Crowdsourced Test Report Aggregation and Summarization

***Abstract*—Crowdsourced testing has been widely adopted to improve the quality of various software products. Crowdsourced workers typically perform testing tasks and report their experiences through test reports. While the crowdsourced test reports provide feedbacks from real usage scenarios, inspecting such a large number of reports becomes a time-consuming yet inevitable task. To improve the efficiency of this task, existing widely used issue-tracking systems, such as JIRA, Bugzilla, and Mantis, have provided keyword-search-based methods to assist users in identifying duplicate test reports. However, on mobile devices (such as mobile phones), where the crowdsourced test reports often contain insufficient text descriptions but instead rich screenshots, these text-analysis-based methods become less effective because the data has fundamentally changed. In this paper, instead of focusing on only detecting duplicates based on textual descriptions, we present CTRAS: a novel approach to leveraging duplicates to enrich the content of bug descriptions and improve the efficiency of inspecting these reports. CTRAS is capable of automatically aggregating duplicates based on both textual information and screenshots, and further summarizes the duplicate test reports into a comprehensive and comprehensible report.**

To validate CTRAS, we conducted quantitative studies using more than 5000 test reports, collected from 12 industrial crowdsourced projects. The experimental results reveal that CTRAS can reach an accuracy of 0.87, on average, regarding automatically detecting duplicate reports, and it outperforms the classic Max-Coverage-based and MMR summarization methods under Jensen Shannon divergence metric. Moreover, we conducted a task-based user study with 30 participants, whose result indicates that CTRAS can save 30% time cost on average without loss of correctness.

I. Introduction

Crowdsourced testing has become a popular mobile application testing method because it can provide feedback from diverse settings, such as devices, location information, network environments, user behaviors, and operating systems. Even though crowdsourced testing has been widely adopted in many commercial testing platforms (such as uTest, Testin, Baidu Crowd Test, Alibaba Crowd Test, and TestIO[1]) and is effective for improving the quality of various software products, inspecting and triaging the overwhelming number of test reports is a time-consuming task in practice.

In the past decades, to improve the efficiency of processing reports, software-engineering researchers have presented many techniques to detect and summarize duplicates. Duplicate-detection techniques aim at assisting developers to filter out duplicate submissions from test-report repositories [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]. Prior research have employed the rich information, such as text descriptions [1, 2, 3, 4, 5, 7, 8, 9, 10] and execution traces [6] to reach this goal. Further, conventional and widely used issue-tracking systems, such as Bugzilla, Jira, and Mantis[2], have provided keyword-search-based features for reporters to query similar reports to reduce duplicate submissions [11, 12]. Also, Rastkar *et al.* presented a test-report summarization technique to assist developers to identify key sentences from test reports to reduce inspection efforts [13].

However, the settings and inherent features of *mobile* crowdsourced testing bring new difficulties into applying these techniques. Zhang *et al.* found that mobile test reports often contain insufficient text descriptions and rich screenshots in comparison with desktop software [14]. Under this situation, while text-analysis-based methods become less effective because of short and inaccurate text descriptions, automatically identifying information from screenshots becomes critical for developers to understand reports. Further, while all of these techniques are built on the assumption that duplicate reports are harmful for software maintenance and aim at filtering out this information, Zimmermann *et al.* and Bettenburg *et al.* empirically found that duplicate reports are helpful for report comprehension and debugging [15, 16].

Thus, in this paper, we propose an approach, named CTRAS, which is capable of leveraging the information of duplicate test reports to assist developers in comprehending test reports. Different from the conventional bug/test-report-processing techniques, instead of discouraging developers from submitting duplicates and filtering them out, our technique aims at leveraging the additional information provided by them, and summarizing both the *textual and image information* from the grouped duplicates to a comprehensive and comprehensible report.

CTRAS automatically detects and aggregates the duplicate reports by measuring the similarity of both the text description and screenshots. Based on the aggregation results, for each duplicate report cluster, it identifies the most informative report, which we call the *master report*, and summarizes the supplementary text and screenshots. These supplementaries are sorted by their weight, and CTRAS generates the final summarized report by combining the master report and supplementaries to provide the developer with a comprehensible overview of each test-report duplicate group.

To validate CTRAS, we conducted both quantitative and

[1]https://www.utest.com, http://www.testin.io, http://test.baidu.com, https://mqc.aliyun.com/crowdtest, https://test.io

[2]https://www.bugzilla.org, https://www.atlassian.com/software/jira, https://www.mantisbt.org

qualitative experiments using more than 5000 test reports collected from 12 mobile applications. The results show that CTRAS can accurately detect and aggregate 87% duplicate reports, by utilizing both text description and screenshot information. CTRAS improves the duplicate report detection and aggregation accuracy by 6%, and by 44% when compared to only text-based, screenshot-based methods. Meanwhile, based on our evaluation result, CTRAS generates more descriptive summaries when compared to classic MCB [17, 18] and MMR [19] summarization methods. Furthermore, we conducted a task-based evaluation involving 30 participants, whose result indicates that CTRAS can save 30% time costs on average without losing correctness.

This paper makes the following contributions:

- We present CTRAS, a mobile crowdsourced test-report processing technique, to assist developers. To the best of our knowledge, CTRAS is the first work that aims at aggregating multiple duplicate test reports into an enriched, summarized report.
- Based on CTRAS, we present a method to detect duplicate test reports that leverages image information. The experiment result shows that the image information can improve the performance of both duplicate detection and aggregation.
- We conducted comprehensive studies on over 5000 test reports from 12 industrial crowdsourced projects to validate CTRAS, and found that CTRAS performed well in terms of clustering accuracy, summarization, efficiency of use, and usability.

II. Background

A. Mobile crowdsourced test reports

In contrast to conventional bug repositories of desktop software applications, bug-report repositories of mobile crowdsourced testing often have higher duplicate ratios, briefer text descriptions, and richer screenshots [14].

High duplicate ratio. Crowdsourced testing is popular in mobile application testing because it enables developers to evaluate the performance of their software products under real usage scenarios. However, in practice, crowd workers are often required to finish crowdsourcing tasks in a given short time, and the number of completed tasks influences the rewards for the crowd workers. As such, crowd workers are less apt to actively filter out duplicates, and they are incentivized to submit as many reports as possible. These factors contribute to crowdsourced testing to contain a higher duplicate ratio than conventional testing.

Short Text and Rich Screenshots. In addition, on almost all mobile devices, images have played a crucial role in sharing, expressing, and exchanging information. End users can easily take a screenshot and the crowd workers tend to describe bugs with a direct screenshot and short descriptions rather than verbose and complex text descriptions, largely due to the ease of taking screenshots and the relative difficulty in typing longer descriptions on mobile virtual keyboards [14]. On mobile platforms, screenshots capture usually well-defined application views, and do not suffer as many of the difficulties of desktop-application screenshots, such as varying resolutions, scaling, occlusion, and window sizes. In this context, the above issues are ameliorated, and the main problems are the prompting of the error-message dialogs, shifting of GUI elements or the fact that some elements are not displayed at all.

B. Motivation

The aforementioned features of mobile crowdsourced test reports, *i.e.,* high duplicate ratio, short text descriptions and rich screenshots, motivate us to propose an approach to leveraging both the text and image information from duplicate reports to enhance developers' understanding of bugs.

Moreover, a common and conventional belief in software-development practice is that the reporting of duplicate test reports is a bad practice and therefore considered harmful. The long and frequent arguments against duplicates are that they strain issue-tracking systems and waste efforts of software maintainers. Thus, based on this argument, prior researchers have proposed many techniques to assist developers in avoiding wasting time on duplicates. However, there are also arguments to the contrary. Zimmerman *et al.* [15] claim that the missing information, such as reproduction steps and environment settings, is one of the most serious problems of test reports of open-source projects. They find that developers often need to spend extra time to interact with reporters to identify the missing information and gain enough understanding of the bug. Bettenburg *et al.* [16] present empirical evidence to show that duplicates provide additional information for describing bugs and this information is helpful for fault localization and fixing. These findings fit the situation of mobile crowdsourced testing, which has been widely adopted in the quality assurance of modern mobile applications.

III. Goals and Preliminary Definitions

In this section, we highlight our main goals and provide definitions and notations that will be used subsequently. Our overall goal is to aggregate duplicate test reports and provide a comprehensible overview of the group. Specifically for mobile software, we seek to provide a technique that is robust and effective in clustering test reports that may contain short text descriptions, but may include screenshots that exhibit failure symptoms.

For a software project with submitted crowdsourced test reports $\mathcal{R}(r) = \{r(S_i, T_i) | i = 0...n\}$, in which, S denotes the screenshots (*i.e.,* images) containing the views that may capture symptoms of the bug being reported, and T denotes the text describing the buggy behavior. Note that each the text description consists of multiple sentences, thus we have $T_i = \{t_{ij} | j = 0...m\}$, in which, t_{ij} denotes the jth sentence in the test report r_i. Similarly, we employ s_{ij} to denote the jth screenshot in the test report r_i.

Summary. The goal of CTRAS is to cluster duplicate reports into groups $\mathcal{G}$, each group of duplicate reports is a subset of $\mathcal{R}$, and then to generate a summary $\mathcal{S}$ for each group in $\mathcal{G}$. In

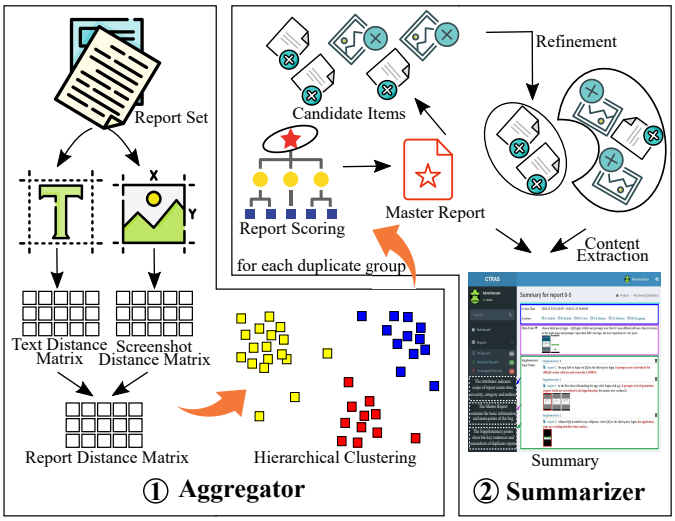

Fig. 1: Process flow of CTRAS

our formulation, we define a *summary* for a group of duplicate test reports as a *master report* and a list of *supplementaries*.

Master Report. In contrast to traditional testing, crowd workers are often inexperienced and unfamiliar with software testing. They may describe a bug from different aspects and in various ways. This fact leads the quality, writing style, and content of these reports to vary widely [20]. Hence, we seek to find one report that provides a relatively comprehensive description of the issue for the group of duplicates. A *master report* is defined as an individual test report $r^\star \in \mathcal{G}$ that is identified as providing the most informative report by some measure (specified in detail in the next section).

Supplementaries. Even though presenting the most informative one or its summary is helpful for developers to build a high-level understanding of the bug, the supplementary information, such as different software and hardware settings, diverse inputs, and various triggers, is critical for developers to gain enough knowledge of debugging and fixing. These supplementary details from other test reports in the duplicate group can provide additional insights to developers in understanding the varying conditions the lead to the issue. Hence, we seek to identify the useful information from the redundant information and summarize them into the comprehensible supplementaries. A *supplementary* is defined as the representative information item, *i.e.,* either text or screenshot, which is taken from $(\mathcal{G} - \{r^\star\})$.

Textual supplementaries are small snippets of text that present information that are not included in the master report, and thus can provide more information and greater context for developers during debugging and triaging. Likewise, image supplementaries are screenshots that differ from those included in the master report.

IV. Approach

We present the architecture of CTRAS in the Fig. 1. CTRAS is composed of two main components: *aggregator* and *summarizer*. The *aggregator* is designed for computing the distance between test reports and aggregating the duplicates into group $\mathcal{G}$. We use hierarchical clustering to accomplish this task. To assist developers understanding the content of the duplicate groups quickly, the *summarizer* first picks a single test report that best exemplifies the group of aggregated test reports, then it supplements the information by gradually extracting *supplementary* information, which contains topics or features uncovered by the *master report*.

A. Aggregator

The *aggregator* first computes the distance of test reports and output the distance matrix. In the distance computation, we adopt the hybrid strategy, proposed by Feng *et al.* [21], to measure the distance between test reports by combining both the image similarity and text similarity. For the textual descriptions, Natural Language Process (NLP) techniques, including tokenization, synonym identification, and keyword vector building, are applied to calculate the text similarity. For the screenshots, the Spatial Pyramid Matching (SPM) [22] technique is adopted to extract the image features and calculate screenshot similarity. The hybrid distance matrix among all reports is built upon the weighted harmonic mean of these two similarities.

Based on the distance matrix, the *aggregator* is capable of measuring the similarity between test reports and further grouping the duplicates. Considering that in practice the number of groups cannot be predicted, we adopt Hierarchical Agglomerative Clustering (HAC), which can determine the number of groups based on a threshold distance value, to group the duplicates [23].

B. Summarizer

The *summarizer* is the core component of CTRAS. For each duplicate test-report groups $\mathcal{G}$, it performs the following three steps to generate a summary to assist developers in forming a comprehensive understanding over all reports in $\mathcal{G}$: (1) identifying the master report $r^\star$, (2) generating supplementaries, and (3) forming and generating final summaries.

To ease explanation, we take a real group containing six test reports in our empirical study as an example, and we illustrate each substeps in Fig. 2. We list the six similar reports in the exemplary group in Fig. 2-a. Note that all of these reports describe the same bug that involves a problem logging into the app via a third-party tool. Each report of this group consists of its basic attributes, such as report names, creation time, as well as its textual description and several screenshots.

1) Master Report Identification: To help the developers concisely understand the topic of the test reports within the group, we identify the *master report* $r^\star$ in the first step. We abstract each test-report group into a graph, within which each node represents an individual report. The weight of edges between two nodes indicates the similarity between these two reports. Thus, we can apply the PageRank [24] algorithm, which can compute a numerical rank score and measure the importance for each node within a weighted graph, on each

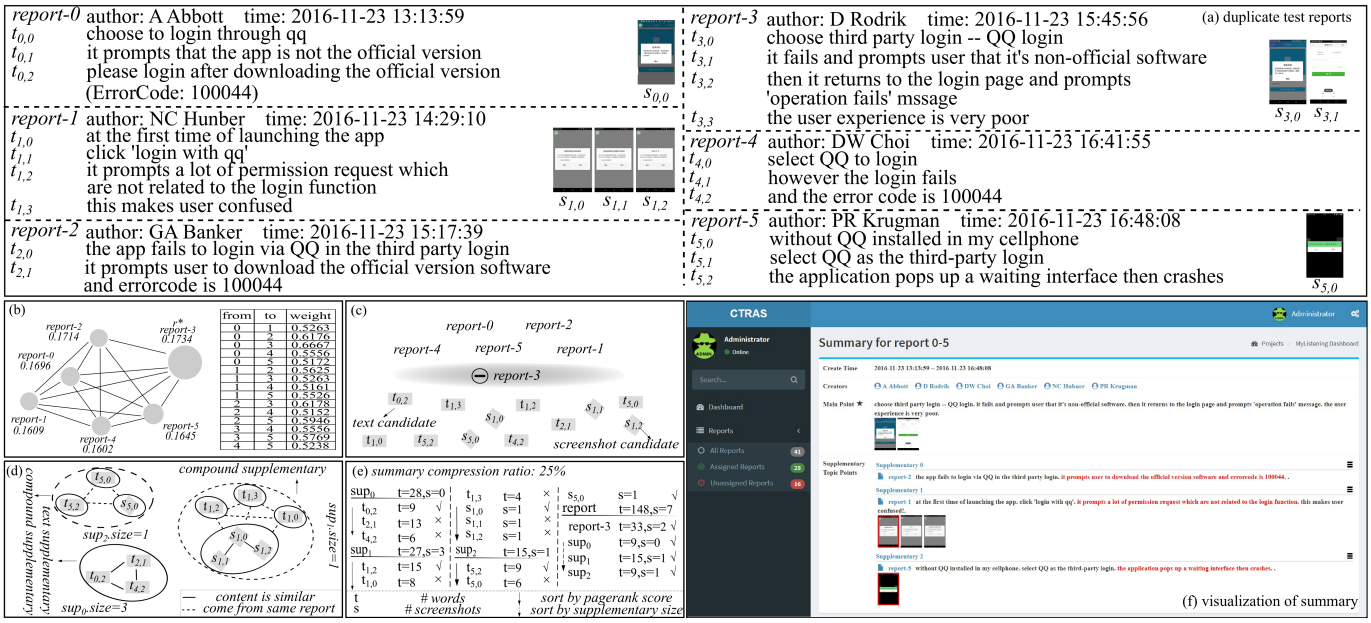

Fig. 2: Example of CTRAS

test-report group. CTRAS identifies the test report having the greatest page rank score as the master report for the group.

Example: The graph representing the exemplary group is illustrated in Fig 2-b, and the table shows the hybrid similarity between each pair of these reports. We compute the weight of each node by applying the PageRank algorithm. We find that *report-3* has the highest weight, and thus it is selected as the master report (labeled with $r^\star$) of these six reports. Through reading the contents of *report-3*, developers can reach a high-level understanding of the whole test report group, *i.e.,*, there is a bug that users can't login the App through QQ social login service as it fails the authentication and is regarded as non-official software.

2) Supplementary Generation: Even though we have identified the *master report* $r^\star$ and helped the developer get the most informative report within the the group $\mathcal{G}$, describing the same bug from other perspectives and providing supplementary materials is critical for developers in fixing the bug properly. Thus, we further analyze the other reports and identify the content that are shared among them as the supplementary points. In this procedure, we perform two substeps for identifying the supplementaries: (1) identifying candidate items from $(\mathcal{G} - \{r^\star\})$; (2) grouping the candidate items to form a *supplementary*.

Candidate Item Identification. Because the sentence is considered to be the immediate integral unit in linguistic theory, a number of prior research efforts that aimed at analyzing test/bug reports to assist developers in understanding the bug descriptions have selected the sentence as the basic unit [13, 25, 26]. We thus also do so accordingly and measure the similarity between two sentences by computing the Jaccard Distance between their keyword vectors. Jaccard Distance is a useful metric to compare the similarity and diversity of two sets, which is shown in Equation 1. In this equation, t and t' denote the keyword sets of two sentences, $|t \cap t'|$ denotes the number of words in the intersection of both sets, and $|t \cup t'|$ denotes the number of words in the union of both sets.

Regarding the screenshot, as we discussed in Section IV-A, we adopt Feng *et al.*'s method and corresponding parameters to identify different screenshots [21]. Given a test-report group $\mathcal{G}$ and its *master report* $r^\star$, to generate supplementaries for $r^\star$, the first step is to identify candidate items, *i.e.,* sentences and screenshots, which are NOT included in $r^\star$ from $(\mathcal{G} - \{r^\star\})$. From the set $(\mathcal{G} - \{r^\star\})$, we extract all singleton items, *i.e.,* individual sentences and screenshots, to get the set of sentences $T = \{t_{ij}\}$ and the set of screenshots $S = \{s_{ij}\}$ of $(\mathcal{G} - \{r^\star\})$. Similarly, we can get the set of sentences and set of screenshots from $r^\star$, and we denote them $T^\star = \{t^\star_j\}$ and $S^\star = \{s^\star_j\}$ respectively. For each sentence whose keyword set is $\{t\}$ in T, if not existing any element in $S^\star$ having the $J(t, t^\star)$ is smaller than the predefined threshold value, we consider it is a candidate sentence for $r^\star$. Similarly, given S and $S^\star$, we can identify the candidate screenshot for $r^\star$.

$$J(t, t') = 1 - |t \cap t'|/|t \cup t'| \quad (1)$$

Example: As shown in Fig. 2-c, we label the text items with rectangles and the screenshot items with diamonds. CTRAS identified eight candidate sentences and four candidate screenshots from the exemplary group. For example, $t_{0,2}$ is a candidate sentence as it supplements the "expected result" information (*i.e.,* "ErrorCode: 100044") for $r^\star$(*i.e., report-3*); $t_{5,0}$ expresses a special case that "without QQ installed" and $s_{5,0}$ is a candidate screenshot for this case.

Candidate Item Refinement. Through candidate item identification, all candidate items, which are not similar to any item in the master report, are identified. However, some of these items may be too brief to understand, or they are similar with each other. Therefore, we refine them into concise and representative *supplementaries*.

The refinement process of CTRAS consists of three substeps: Step 1 clusters similar candidate items; Step 2 provides an additional clustering of the candidate clusters; and Step 3 weights the candidates within the clusters to identify the

most representative among them. In Step 1, we group similar candidate items to remove redundancy and improve the conciseness of the supplementary. In this step, we apply hierarchical agglomerative clustering on the set of candidate sentences to form some candidate sentence clusters. Similarly, we can apply the same strategy on the candidate screenshot set and get the candidate screenshot clusters, using approach Spatial Pyramid Matching described in Section IV-A. Moreove, we record the origin information for each candidate, so that we can map candidates to reports and vice versa. This information is not only useful for further aggregating candidate sentence clusters in the next sub-step but also helpful for developers to track back original reports in practice.

In Step 2, we further group the candidate clusters from Step 1. The purpose for this step is to restore context. This extra level of grouping is useful particularly for singleton clusters (clusters containing only a single candidate item due to the fact it contains distinct information). We merge the clusters based on the origin information: clusters that contain candidates originating from much of the same test reports are clustered. We define the distance between two candidate item clusters t and s as Equation 2, in which, each cluster can be either candidate sentence cluster or candidate screen cluster and the $\Phi(t)$ represents the set of test reports that contributed to the candidate item cluster t. Based on the Equation 2, candidate item clusters are aggregated into *supplementaries* when the distance between them is smaller than threshold value θ. *Raw supplementaries* should not be presented to the end-user because they contain too much redundancy.

$$D(t,s) = 1 - |\Phi(t) \cap \Phi(s)| \,/\, |\Phi(t) \cup \Phi(s)| \qquad (2)$$

In Step 3, we identify the most representative candidate items in each supplementary cluster. Based on our definition of sentence similarity and screenshot similarity, we abstract all sentences and screenshots within a supplementary into a weighted graph respectively and employ the similarity value as the weight of edges. Given these two weighted graphs, we apply the PageRank algorithm and obtain the PageRank score for each of the node, *i.e.,* sentences and screenshots. These weights will be used within the next phase of content extraction to highlight the most relevant and representative information for each supplementary.

Example: Fig. 2-d displays the refinement result of all *candidate items*: three *candidate sentences* (*i.e.,* $t_{0,2}$, $t_{2,1}$ and $t_{4,2}$) are grouped together because they contain "100044 error code"; *candidate clusters* $\{t_{5,0}\}$, $\{t_{5,2}\}$ and $\{s_{5,0}\}$ are grouped because they belong to *report-5*. Particularly, the size of *supplementary-0* is 3 as its content comes from three reports.

3) Content Extractor: Based on *master report* and *supplementaries*, we can further refine them and generate a concise final summary.

In many textual summarization techniques (*e.g.,* [13, 25, 27, 28]), the compression ratio K controls the conciseness of the final summary. In previous works, compression ratio is computed as the ratio of the number of selected keywords to the number of total keyword within the original document. However, because CTRAS aims at generating summary over both text and screenshots, we extend the classic definition. For the text, we define the compression ratio as the ratio of the number of unique selected word to the total number of unique word within the supplementary. Similarly, for the screenshot, the compression ratio is the ratio of the number of selected screenshots to the total number of screenshots within the supplementary. We weight text and screenshot equally and thus utilize the mean value of these two compression ratios as the compression ratio for the whole summary.

To generate the final summary, we first include master report, and then list all *supplementaries* sorted by the number of test reports that contributed to them in descending order. For each *supplementary*, we iteratively select the sentence or screenshot based on the weights (computed in Step 3 of the candidate refinement phase) and include them into the summary, until reaching the summary compression ratio set by the user.

Example: The detail summarizing process is shown in Fig. 2-e. We take the *supplementary-0* as sample. It contains 28 keywords and 0 screenshots. At the beginning of summarization, the sentence $T_{0,2}$ is selected as it has the highest PageRank score, then the summary contains 9 keywords and the compression ratio has reached the limit (*i.e.,* the compression ratio is $9/28 > 0.25$), the summarization process ends.

C. Implementation

We have implemented a web-based test-report-management tool, which not only provides test-report summarization but also extends a number of classic test-report-management functions, such as automatic duplicate detection, a project-report dashboard, keyword searching, bug triaging, and best fixer recommendation.

We present the screenshot of its summary visualization page in Fig. 2-f, which shows visualization result of the final summary from the exemplary duplicate group. At the top of this page, we present attribute information about the aggregate report (such as the set of all crowd testers who submitted reports in this set, bug category, severity). And then, we show information from the master report, which aims at assisting users to get an overview understanding of all duplicates. Below the master-report pane, topics of supplementary are listed. We highlight the representative sentences, phrases, and screenshots of each supplementary, which enables end users to understand the main topic of this supplementary at a glance. Further, end users can view the details, including all sentences, screenshots and original reports contributed the supplementary, by clicking these supplementary topics. Using on this tool, we conducted comprehensive studies to validate our approach, described in the next section.

V. Experiment

To assess the performance of CTRAS in achieving its goals, *i.e.,* to assist developers in (1) processing test reports, (2) providing comprehensive and comprehensible summaries, and (3) saving efforts, we conduct mixed evaluations to answer the following three research questions:

RQ1. Effectiveness of Duplicate Aggregator. Can the *aggregator* accurately group duplicates? To what extent do the screenshots improve the accuracy of detecting and aggregating duplicate reports?

RQ2. Effectiveness of Summarizer. To what extent can the *summarizer* refine the informative and non-redundant content from the duplicates?

RQ3. Effectiveness of CTRAS. Can CTRAS help developers save time costs in inspecting mobile crowdsourced test reports without loss of accuracy?

Both RQ1 and RQ2 are designed to evaluate the effectiveness of the *Aggregator* and *Summarizer* components of CTRAS. Because identifying and aggregating duplicate reports are foundational steps of correctly leveraging the information from test reports, RQ1 aims at evaluating the accuracy of CTRAS in detecting duplicates and revealing the effectiveness of employing the screenshot information to aggregate the duplicates. Also, identifying the critical, complementary, and non-redundant information from the large volume of information plays an important role in helping developers to understand and fix the bugs. Thus, RQ2 aims at evaluating the effectiveness of the *summarizer* regarding the metrics of information theory. Further, although RQ1 and RQ2 present quantitative and theoretical evaluations, understanding the practical performance of CTRAS is critical. Thus, we design RQ3, which is a task-based user study, to investigate the efficiency improvement as well as reporting any accuracy loss.

As discussed in Section IV, several fundamental parameters may influence the performance of CTRAS. We provide our parameterizations for all three experiments to assist verifiability and repeatability. Feng *et al.* suggested that the β should be adjusted based on different tasks, and given previous researchers, Bettenburg *et al.* [15], found that the textual information (*e.g.,* description, observed and expected behavior, reproduction steps) is capable of providing more accurate description than screenshots for developers in debugging, we set $\beta = 5$ to weight textual descriptions more. Also, there are two fundamental parameters of the HAC algorithm: the linkage type, which defines the method of calculating the distance between clusters, and the threshold γ for terminating the clustering. We choose the single-linkage type because it makes the HAC to solely focus on the area where the two clusters come closest to each other and ignore distant parts [29], which fits the goal of CTRAS well. We set the γ to 0.5 that is the medium value of the scale of the distance between test reports. Further, we apply strict combination strategy by setting $\theta = 0.2$ that we defined in Section IV-B2 to group candidate item clusters. In the study of RQ1 and RQ3, we set the compression ratio $K = 0.25$, which is considered to be a proper value in the paper[13, 27].

TABLE I: Statistical Information of Testing Applications

	Name	Version	Category	$\|R\|$	$\|S\|$	$\|R_s\|$	$\|D\|$
$p1$	CloudMusic	2.5.1	Music	157	259	62	45
$p2$	Game-2048	3.14	Games	210	219	164	96
$p3$	HW Health	2.0.1	Health	262	327	201	109
$p4$	HJ Normandy	2.12.0	Education	269	436	241	123
$p5$	MyListening	1.6.2	Education	473	418	306	128
$p6$	iShopping	1.3.0	Shopping	290	508	150	83
$p7$	JayMe	2.1.2	Social	1400	1997	1168	678
$p8$	JustForFun	1.8.5	Photo	267	112	76	141
$p9$	Kimiss	2.7.1	Beauty	79	58	48	31
$p10$	Slife	2.0.3	Health	1346	2238	1124	885
$p11$	Tuniu	9.0.0	Travel	531	640	418	236
$p12$	Ubook	2.1.0	Books	329	710	88	108
$total$				5613	7922	4046	2663

A. Dataset Description

To produce the dataset for our evaluation, we utilized the results of the national software-testing contest[3], which simulated crowdsourced testing of several popular mobile applications across multiple domains (including games, education, social media, and so on). The contestants of the contest were required to test the applications and report bugs in four hours. They could write descriptions and take screenshots to document their testing procedures and the unexpected behavior of applications. This contest attracted 4000 participants and generated over 5000 test reports. More than 10 professional testers and members of the organizational committee manually labeled and evaluated the quality of these reports. The detailed information of the dataset is shown in Table I, in which, $|R|$ denotes the number of reports, $|S|$ denotes the number of screenshots, $|R_s|$ denotes the number of reports that contains at least one screenshot and $|D|$ denotes the number of duplicates.

B. RQ1. Effectiveness of Duplicate Aggregator.

1) Methods: While a number of classic duplicate test-report-detection methods only focus on the textual description to measure the similarity between reports [1, 2, 3, 4, 5, 7, 8, 9, 10], CTRAS employs both textual description as well as screenshots to assist detecting duplicates. Thus, to answer the RQ1, we have the following three methods:

- **CTRAS.** Our duplicate detection method which employs both textual information and screenshots. In this method, the distance between two reports is calculated based on the balanced distance equation.
- **CTRAS-TXT.** The duplicate detection method employs only textual information. In this method, the distance is calculated based on only textual distance.
- **CTRAS-IMG.** The duplicate detection method employs only screenshot information. In this method, the distance is calculated based on only screenshot distance.

2) Evaluation Metrics: To measure the performance of these three methods, we employ three classic metrics for evaluating clustering: Homogeneity, Completeness, V-Measure [30]. Taking the classes set C and clusters set K as reference, we define the contingency table $\mathcal{A} = \{a_{ij} | i = 1, ..., n; j = 1, ..., m\}$, where a_{ij} denotes the number of test reports that belongs to both c_i and k_j.

[3]http://www.mooctest.org/cst2016/index_en.html

Homogeneity reflects the extent to which each cluster contains only members of a single class. It can be calculated via $h = 1 - H(C|K)/H(C)$, where

$$H(C|K) = -\sum_{j=1}^{|K|}\sum_{i=1}^{|C|}\frac{a_{ij}}{N}\cdot log\frac{a_{ij}}{\sum_{k=1}^{|C|}a_{kj}} \tag{3}$$

$$H(C) = -\sum_{i=1}^{|C|}\frac{\sum_{j=1}^{|K|}a_{ij}}{n}\cdot log\frac{\sum_{j=1}^{|K|}a_{ij}}{n} \tag{4}$$

Completeness is a symmetrical criterion of homogeneity, which measures the extent to which all members of a given class are assigned to the same cluster. It can be calculated via $c = 1 - H(K|C)/H(K)$, where

$$H(K|C) = -\sum_{i=1}^{|C|}\sum_{j=1}^{|K|}\frac{a_{ij}}{N}\cdot log\frac{a_{ij}}{\sum_{k=1}^{|K|}a_{ik}} \tag{5}$$

$$H(K) = -\sum_{j=1}^{|K|}\frac{\sum_{i=1}^{|C|}a_{ij}}{n}\cdot log\frac{\sum_{i=1}^{|C|}a_{ij}}{n} \tag{6}$$

V-measure is the harmonic mean of homogeneity and completeness. It is widely used as the measure of the distance from a perfect clustering. In our paper, a higher V-Measure score indicates a better duplicate detection and aggregation result, which is calculated by the Equation 7.

$$v = 2\cdot(h\cdot c)/(h+c) \tag{7}$$

C. RQ2. Effectiveness of Summarizer

1) Methods: To investigate the theoretical effectiveness of the summarizer of CTRAS, we compared its performance with two classic summarization methods: Max-Coverage-based (MCB) [17, 18] and Maximal Marginal Relevance (MMR) [19].

- **CTRAS.** Our summarization method that generates summarized report under ratio K.
- **MCB.** The Max-Coverage-based method is a greedy algorithm. *MCB* iteratively selects the test report with maximal coverage score and inserts it into the summarization until K is met. The original definition of coverage score in paper [17, 18] refers the ratio of the number of selected conceptual units to the total number. In this experiment, we have two kinds of conceptual units, *i.e.,* keywords and screenshots. Thus, we define the coverage score as the mean value of the keyword coverage score and screenshot coverage score.
- **MMR.** The MMR method is a typical method for summarizing multiple topically related documents, which employs keywords that have the highest frequency to build a query [31]. This query is used to select the document from a set based on the maximum-marginal-relevance strategy, which selects the one having the largest distance from the selected set while being relevant to query in each step. In our implementation, we adopt the same idea and construct the query with keyword and screenshot having the highest frequency. We employ the distance between test reports, which we defined in Section IV-A, as the distance measurement to implement the maximum-marginal-relevance strategy.

Note that we define the compression ratio of the final summary as the mean value of the text compression ratio and screenshot compression ratio (see Section IV-B3) — this strategy is also applied in this experiment.

2) Evaluation Metrics: We adopt a fully automatic evaluation method for content selection: the **Jensen Shannon divergence** (JS divergence), which has been shown to be highly correlated with manual evaluations and sometimes even outperforms standard Recall-Oriented Understudy for Gisting Evaluation (ROUGE) scores [32].

JS divergence employs the probability distribution of words to measure the distance between documents. A good summary is expected to have low divergence with the original document. In this paper, we calculate the JS divergence of textual information and screenshots respectively.

The JS divergence is represented in Equation 8, in which, P and Q denote the probability distributions of word $\mathcal{G}$ and summary $\mathcal{S}$, respectively. We entirely adopt the recommended parameter settings in [32], *i.e.,*, $A = (P+Q)/2$ denotes the mean distribution of P and Q, C denotes the frequency of keyword ω, N is the sum of frequencies of all keywords, $B = 1.5|V|$ where V denotes the text corpus, and δ is assigned to 0.0005 to perform a small smoothing. JS_S is defined in a similar manner.

$$JS_T(P||Q) = (D(P||A) + D(Q||A))/2 \tag{8}$$

where

$$D(P||Q) = \textstyle\sum_\omega p_P(\omega) log_2 \frac{p_P(\omega)}{p_Q(\omega)}$$
$$p(\omega) = (C+\delta)/(N+\delta * B)$$

After JS_T and JS_S are calculated, we utilize their harmonic mean as the measure of these summarization methods.

D. RQ3. Effectiveness of CTRAS

Although RQ2 evaluates the theoretical performance of CTRAS, we also seek to understand its practical performance for real users. In [13], Rastkar *et al.* designed a task-based user study to investigate whether the generated summaries can help developers in processing test reports. In their study, participants were asked to read a new test report and a list of potential duplicated reports, which were presented under their *original* or *summary* format, and then determine for each whether it was duplicated with the new test report or not. Considering both Rastkar *et al.*'s work and ours share the same goal, we adopt the task-based user study to answer the RQ3. However, because their work is designed to produce a summary for a single test report while CTRAS generates a summary for multiple test reports, we adjust the duplicate test-report-detection task into duplicate test-report clusterization tasks in our study.

For our study, we utilized a modified version of the web-based CTRAS tool to assist participants to cluster test reports. Our participants are given a set of test reports (see Table I), and

optionally a set of summaries, and asked to group duplicate test reports (*i.e.,* test reports describing the same bugs). Our hypothesis is that CTRAS can help developers reach multi-perspective understanding of the bug and thus identify the duplicates more efficiently without loss of accuracy. The rationale of this study design is straightforward: if the summary generated by CTRAS failed to provide *sufficient* and *correct* knowledge for participants to understand the bug, they cannot correctly identify the duplicate report that are describing the same bug.

1) Study Setting: We recruited 30 second-year master students majoring in computer science or software engineering as participants of this study. All of these 30 participants have at least 5 years programming experience but have no experience using any of these experimental applications.

We select 5 applications, *i.e., MyListening (p5)*, *iShopping (p6)*, *Kimiss (p9)*, *Tuniu (p11)*, *Ubook (p12)* as subject programs. The categories of these applications are diverse and the number of reports vary from 79 to 531, thus we believe these applications are representative.

We randomly divide the 30 participants into 3 groups. In the study, these three groups are provided different reference materials:

- **Group A (Control Group):** The participants of this group are only provided with the original test report. There are no supportive materials for the participants of this group.
- **Group B (CTRAS):** The participants of group B are provided with the original test report and the summary that is generated by CTRAS. Note that, for this group, the summarizer works on the fully-automated aggregator, which groups the test reports based on the both image and text similarity.
- **Group C (Golden):** The participants of group C are provided with the original test report and summaries that are generated by the summarizer of CTRAS working on the ground-truth clustering results (as manually determined by the professional developers, described in Section V-A). Because the quality of summaries is influenced not only by the summarization algorithm but also by the duplicate aggregation algorithm, we set up this group to evaluate the gap between the performance of CTRAS and the perfect situation.

Within each group (10 participants, each), every subject application (5 software applications) is assigned to two participants. Participants are required to manually cluster these original test reports independently — without any collaboration. The modified version of CTRAS shows summaries without showing any information that reveal test report identities — simply showing the summarized sentences and screenshots that describe bugs. This version of CTRAS also provides keyword search and keyword filtering.

We employ CTRAS to generate summaries under the predefined condition of group B and C, and then provide these summaries to the participant of corresponding groups as reference material.

TABLE II: Details of the summarization result in RQ3

		p5		p6		p9		p11		p12	
		B	C	B	C	B	C	B	C	B	C
#summary		98	97	35	63	15	25	145	182	64	96
#sentences	mean	3.12	4.13	2.33	3.15	2.25	6.00	4.51	4.80	3.85	5.00
	std	1.95	3.77	2.01	1.96	1.85	2.10	3.46	2.63	3.37	7.93
#screenshots	mean	1.73	2.35	3.38	4.05	1.25	4.40	3.51	3.69	7.60	9.42
	std	1.84	2.73	5.38	3.11	1.09	3.38	3.80	3.30	10.69	21.92

The details of summarization results is illustrated in Table II. For each subject application and group (B & C), we show the number of summaries and their mean number of sentences and screenshots.

2) Evaluation Metrics: We evaluate CTRAS based on three aspects: efficiency, accuracy, and satisfaction.

- **Efficiency.** We adopt the average completion time for each report as the evaluation metric of efficiency.
- **Accuracy.** Using the ground truth data described in Section V-A, we determined the accuracy of the participant's inspection by utilizing V-measure metric (see definition in Section V-B2).
- **Satisfaction.** The satisfaction of summary is measured upon the qualitative feedback from the questionnaire, which is shown in Table III. Particularly, we present the questionnaire only to participants of group B and C.

VI. Result Analysis

In this section, we present the experimental results to answer the three research questions.

A. Answering RQ1: Effectiveness of the Duplicate Aggregator

We present the homogeneity (H), completeness (C) and V-Measure (V) results of CTRAS and the two baseline techniques in Table IV.

On average, CTRAS achieves 0.87 V-Measure score, while these two baseline techniques, *i.e.,* CTRAS-TXT and CTRAS-IMG, obtain 0.81 and 0.60 respectively.

Further, CTRAS outperforms these two baseline techniques over all subject projects except the *"Slife"(p10)*. We investigated the content of test reports of subject projects. We found the application *Slife* is a daily activity tracker, which is designed for tracking the health data of users' daily activity. Even though its operation is simple, the testing procedure, which requires a number of activities beyond the regular operations, becomes relatively difficult. Given the fact that our test reports come from the contest which requires participants to finish the tasks in a short time (4 hours), we speculate that the test reports of *Slife* could only reveal simple bugs, and as such and their text descriptions were accurate. Thus, CTRAS-TXT obtains the highest homogeneity score, which results to relatively higher V-Measure score in comparison with CTRAS.

Summary: The high V-Measure score indicates that the *duplicate aggregator* is capable of accurately detecting and aggregating duplicate reports. In comparison with the classic text-only-based strategies, the screenshot information is able to improve the performance of detecting duplicates (in 11 out of 12 subject applications).

TABLE III: Interview Questions in RQ3

1. On a scale of 1–5 with 5 being the most positive, how would you describe the overall performance of the summary in assisting your clusterization task?
2. Does the master report reflect the topic of the summary; if yes, how does it reflect it?
3. Is there additional information in the supplementaries that helped you cluster test reports? If so, describe it.
4. Which type of information is more important for you in inspecting reports: textual descriptions or screenshots? Why?

TABLE IV: Evaluation Results for the Duplicate Aggregator: RQ1

		$p1$	$p2$	$p3$	$p4$	$p5$	$p6$	$p7$	$p8$	$p9$	$p10$	$p11$	$p12$	avg
H	CTRAS	**0.967**	**0.991**	**0.990**	**0.991**	**0.957**	**0.948**	**0.958**	0.865	**0.995**	0.862	**0.994**	**0.967**	**0.957**
	TXT	0.679	0.818	0.918	0.930	0.874	0.857	0.931	**0.878**	0.900	**0.932**	0.932	0.698	0.862
	IMG	0.444	0.416	0.724	0.789	0.490	0.507	0.722	0.222	0.386	0.650	0.703	0.212	0.522
C	CTRAS	**0.778**	**0.782**	**0.876**	0.855	**0.925**	**0.904**	**0.851**	**0.493**	**0.845**	**0.657**	0.877	**0.774**	**0.801**
	TXT	0.660	0.755	0.872	0.856	0.909	0.903	0.839	0.426	0.834	0.628	**0.880**	0.713	0.773
	IMG	0.748	0.682	0.863	**0.863**	0.883	0.874	0.821	0.318	0.693	0.602	0.864	0.525	0.728
V	CTRAS	**0.862**	**0.874**	**0.929**	**0.918**	**0.941**	**0.926**	**0.901**	**0.628**	**0.914**	0.745	**0.932**	**0.860**	**0.869**
	TXT	0.670	0.785	0.894	0.891	0.891	0.879	0.883	0.574	0.866	**0.750**	0.905	0.705	0.808
	IMG	0.557	0.517	0.788	0.824	0.630	0.642	0.768	0.262	0.496	0.625	0.775	0.303	0.599

B. Answering RQ2: Effectiveness of the Summarizer

The results of RQ2 are shown in Fig. 3, for all subject application with varying compression ratios. We note that regardless of the compression ratio, CTRAS generally outperforms MCB and MMR methods in all projects except "JustForFun."

Through further investigation, we found that "JustForFun" is an image editing and sharing application. Thus, the screenshots are largely composed of user content (*i.e.,* their photos) instead of more standardized activity views, so most screenshots have a large distance from each other, which causes them to be categorized as independent supplementaries. This causes a decrease in JS divergence. In addition, as the summarization ratio increases, the score of the JS divergence decreases except for the CTRAS result on project "Game-2048," which is caused by the fact that there is only one gaming interface. That is to say the overwhelming majority of screenshots are similar. Few screenshots can represent the whole corpus, thus the JS divergence is smaller under lower summarization ratio.

Summary: For most projects, CTRAS is more effective than classic summarization methods: MCB and MMR.

C. Answering RQ3: Effectiveness of CTRAS

1) Efficiency & Accuracy: Table V shows the average test report inspection time cost, per test report, for group A (*i.e.,* control group), B (*i.e.,* CTRAS) and C (*i.e.,* golden). According to Table V, the average completion time cost of each report are 19.32 and 20.58 seconds, respectively for group B and C, which saves 30.0% and 25.5% compared with group A (27.63 seconds); and the average V-measure scores of group A, B, and C are 0.9071, 0.9316, and 0.9400 respectively, which shows that group B and group C improve 2.7% and 3.6% accuracy compared with group A. This result indicates that with the help of summarization, people can substantially save their time in duplicate test report clustering work not only without loss of accuracy, but even with slight improvement.

In addition, surprisingly, group B cost less time than group C on average. We investigated the details of the summarization result that is presented in Table II. We find that that CTRAS performs a more strict duplicate aggregation than the professional developers, which leads the number of clusters for group B to be smaller than group C. As such, participants of group B generally have fewer summaries to reference in the inspection procedure. Thus, in comparison with group C, group B can save some time-cost at the expense of loss of accuracy.

TABLE V: Task Evaluation Results: RQ3

	A	B	C
completion time (s)	27.6293	19.3179 (30.0%)	20.5754 (25.5%)
v-measure	0.9071	0.9316 (2.7%)	0.94 (3.6%)

2) Satisfaction: The participants' satisfaction rating on average was high: 4.1 on a 1–5-point scale. More subjectively, the semi-structured interviews produced qualitative results. All participants thought that the *master report* can reflect the topic of summary, and it "*helps them get a general idea of the summary*", "*instructs the granularity of clustering*", 18 participants (90%) mentioned supplementaries contained additional information which "*is clear and coherent*", "*can be used as valuable reference when it comes to uncertain condition*" and "*provides detailed operation steps*."

Many participants mentioned text was more useful, which supports our strategy of setting distance calculation parameters described in Section V. Some participants stated that screenshots "*are open to various interpretations*" and "*can't tell where's the problem*." Moreover, some suggestions for improvement were proposed, such as "*the description is not well structured*," and "*highlighting important parts of screenshots*."

D. Threats to Validity

1) Subject Program Selection: The first threat is related to the generality of CTRAS. We evaluated our approach on 12 projects, all of which are Android applications, thus it is unclear whether CTRAS can achieve similar results on other projects from Android and other mobile platforms (*e.g.,* iOS). However, the categories of our projects vary widely, such as Music & Audio, Games, Health & Fitness, Education, Travel & Local, and so on. Therefore, we believe the experiment result can indicate the usefulness of our method.

2) Natural Language Selection: All the crowdsourced test reports utilized in this paper are written in Chinese, which may affects the generalization of CTRAS. However, the purpose of our method is to generate comprehensive summaries by leveraging the information in duplicated reports, and its key part is to measure the similarities between reports utilizing textual description and screenshots. In the aspect of textual descriptions, the similarities are effected by the keyword-corpus extraction methods, and NLP researchers have pro-

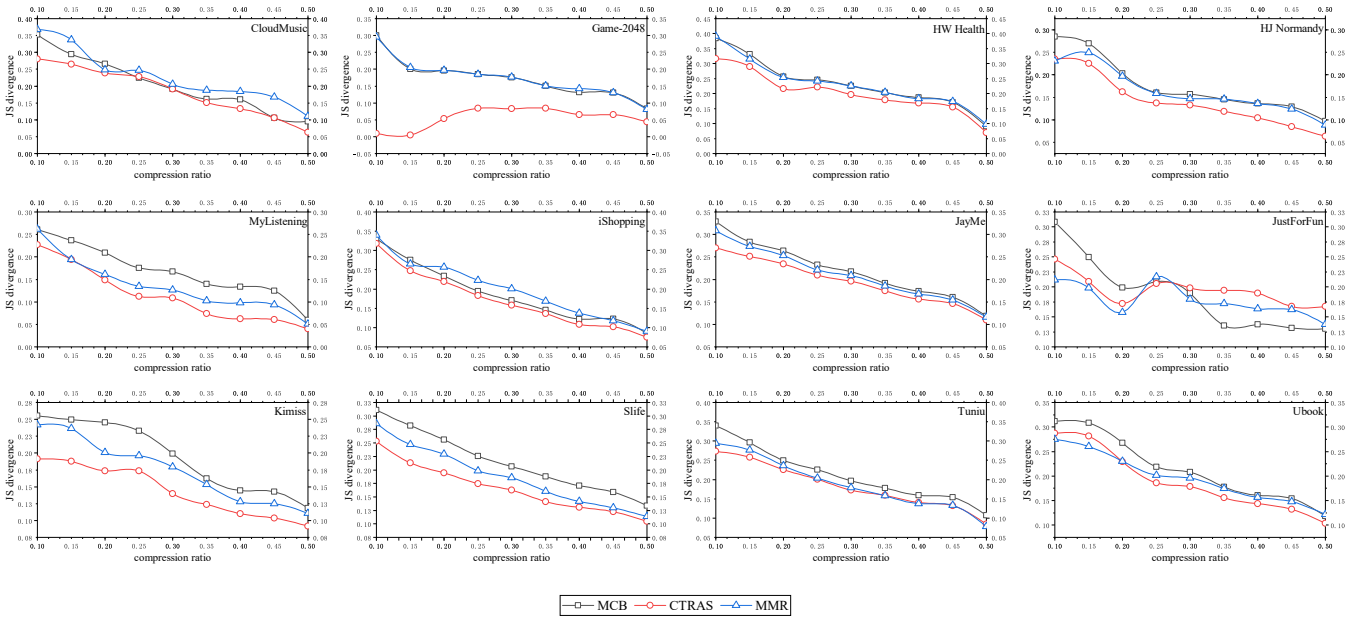

Fig. 3: Evaluation Results for the Summarizer: RQ2 (lower is better)

posed many relatively mature approaches targeting different languages.

VII. Related Work

A. Duplicate Report Detection

Duplicate report detection is the problem of verifying whether a new test report is a duplicate of existing reports, it has been well studied by many researchers. Previous duplicate detecting approaches are mainly based on natural language processing techniques [1], machine learning techniques [2, 3, 4, 5], information retrieval techniques [6, 7, 8, 9]. Lately, Deshmukh *et al.* [10] sought to apply deep learning on detecting duplicate reports, the result indicated their method can outperform the state-in-art approaches.

The duplicates also exists in crash reports besides test reports, to facilitate managing crashes efficiently. Dang *et al.* [33] considered a novel bucketing method called ReBucket that clusters the crash reports based on call stack matching using the Position Dependent Model (PDM). Jiang *et al.* [34] applied the clustering technique on crowdsourced test reports. They proposed TERFUR to aggregate multiple redundant test reports into clusters to reduce the number of report that need to be inspected manually.

However, none of these approaches leverages the screenshot information on duplicate report detection problem, and the goal of these works is to filter out the duplicates, whereas our goal is to take advantage of duplicates to help people understand bugs more comprehensively.

B. Bug Report Summarization

There are several works discussing the problem of bug report summarization, which are resolved in either a supervised or unsupervised way. Rastkar *et al.* [13, 27] developed a supervised learning classifier to judge whether a sentence should be included in the summary, and found that the classifier trained specifically on bug reports outperformed existing conversation-based classifiers. Jiang *et al.* [35] leveraged the authorship characteristics to assist bug report summarization. The intuition is that, given a new bug report created by contributor A, the classifier trained over annotated bug reports by A is highly likely to generate better summaries than classifiers trained over annotated bug reports by others. Jiang *et al.* [28] proposed a PageRank-Based Summarization Technique (PRST), which utilized the information in a master report and associated duplicates to summarize the master report.

To address the problem that supervised approaches require manually annotated corpora and generated summaries may be biased towards training data, Mani *et al.* [26] applied four well known unsupervised summarization algorithms to bug report summarization. Lotufo *et al.* [25] proposed a hypothetical model of user's bug report reading process and utilized this model to rank sentences by its probability of being read and include sentences with the highest probabilities into summary.

In contrast, instead of summarizing individual bug reports, our approach utilizes the information from all duplicate reports to provide a summary view of a set of related reports.

VIII. Conclusion

The problem of diagnosing the overwhelming number of test reports has been a fundamental challenge for crowdsourced testing. To alleviate this problem, in this paper we present CTRAS, a novel approach for aggregating and summarizing crowdsourced test reports. CTRAS leverages the duplicate reports to assist professional testers to manage and understand crowdsourced test reports. It overcomes several shortcomings by: (1) aggregating duplicates to enable batch processing, (2) summarizing the supplementary topics from duplicates to facilitate developer comprehension of the reports. The evaluation result reveals that CTRAS is capable of assisting people's detecting and triaging crowdsourced test reports.

REFERENCES

[1] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *International Conference on Software Engineering*, 2007, pp. 499–510.

[2] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo, "A discriminative model approach for accurate duplicate bug report retrieval," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 45–54.

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

[4] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *Proceedings of the 10th Working Conference on Mining Software Repositories*. IEEE Press, 2013, pp. 183–192.

[5] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014, pp. 308–311.

[6] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*. IEEE, 2008, pp. 461–470.

[7] A. Sureka and P. Jalote, "Detecting duplicate bug report using character n-gram-based features," in *Software Engineering Conference (APSEC), 2010 17th Asia Pacific*. IEEE, 2010, pp. 366–374.

[8] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 70–79.

[9] A. Hindle, A. Alipour, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection and ranking," *Empirical Software Engineering*, vol. 21, no. 2, pp. 368–410, 2016.

[10] J. Deshmukh, S. Podder, S. Sengupta, N. Dubash *et al.*, "Towards accurate duplicate bug retrieval using deep learning techniques," in *Software Maintenance and Evolution (ICSME), 2017 IEEE International Conference on*. IEEE, 2017, pp. 115–124.

[11] A. Hindle, "Stopping duplicate bug reports before they start with continuous querying for bug reports," PeerJ Preprints, Tech. Rep., 2016.

[12] M. S. Rakha, C.-P. Bezemer, and A. E. Hassan, "Revisiting the performance of automated approaches for the retrieval of duplicate reports in issue tracking systems that perform just-in-time duplicate retrieval," *Empirical Software Engineering*, pp. 1–25, 2017.

[13] S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," *IEEE Transactions on Software Engineering*, vol. 40, no. 4, pp. 366–380, 2014.

[14] T. Zhang, J. Chen, X. Luo, and T. Li, "Bug reports for desktop software and mobile apps in github: What is the difference?" *IEEE Software*, 2017.

[15] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schroter, and C. Weiss, "What makes a good bug report?" *IEEE Transactions on Software Engineering*, vol. 36, no. 5, pp. 618–643, 2010.

[16] N. Bettenburg, R. Premraj, and T. Zimmermann, "Duplicate bug reports considered harmful ... really?" in *IEEE International Conference on Software Maintenance*, 2008, pp. 337–345.

[17] E. Filatova and V. Hatzivassiloglou, "A formal model for information selection in multi-sentence text extraction," in *Proceedings of the 20th international conference on Computational Linguistics*. Association for Computational Linguistics, 2004, p. 397.

[18] H. Takamura and M. Okumura, "Text summarization model based on maximum coverage problem and its variant," in *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*. Association for Computational Linguistics, 2009, pp. 781–789.

[19] J. Carbonell and J. Goldstein, "The use of mmr, diversity-based reranking for reordering documents and producing summaries," in *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1998, pp. 335–336.

[20] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced testing," in *Joint Meeting*, 2015, pp. 225–236.

[21] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *Ieee/acm International Conference on Automated Software Engineering*, 2016, pp. 202–213.

[22] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer vision and pattern recognition, 2006 IEEE computer society conference on*, vol. 2. IEEE, 2006, pp. 2169–2178.

[23] A. K. Jain and R. C. Dubes, *Algorithms for Clustering Data*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[24] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[25] R. Lotufo, Z. Malik, and K. Czarnecki, "Modelling the hurried bug report reading process to summarize bug reports," *Empirical Software Engineering*, vol. 20, no. 2, pp. 516–548, 2015.

[26] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: approach for unsupervised bug report summari-

zation," in *ACM Sigsoft International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[27] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: a case study of bug reports," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 2010, pp. 505–514.

[28] H. Jiang, N. Nazar, J. Zhang, T. Zhang, and Z. Ren, "Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates," *International Journal of Software Engineering and Knowledge Engineering*, vol. 27, no. 06, pp. 869–896, 2017.

[29] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press, 2008, vol. 39.

[30] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure," in *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, 2007.

[31] J. Goldstein, M. Kantrowitz, V. Mittal, and J. Carbonell, "Summarizing text documents: Sentence selection and evaluation metrics," in *In Research and Development in Information Retrieval*, 1999, pp. 121–128.

[32] A. Louis and A. Nenkova, "Automatically evaluating content selection in summarization without human models," in *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*. Association for Computational Linguistics, 2009, pp. 306–314.

[33] Y. Dang, R. Wu, H. Zhang, D. Zhang, and P. Nobel, "Rebucket: A method for clustering duplicate crash reports based on call stack similarity," in *International Conference on Software Engineering*, 2012, pp. 1084–1093.

[34] H. Jiang, X. Chen, T. He, Z. Chen, and X. Li, "Fuzzy clustering of crowdsourced test reports for apps," *ACM Transactions on Internet Technology (TOIT)*, vol. 18, no. 2, p. 18, 2018.

[35] H. Jiang, J. Zhang, H. Ma, N. Nazar, and Z. Ren, "Mining authorship characteristics in bug repositories," *Science China Information Sciences*, vol. 60, no. 1, p. 012107, 2017.