Estimate the Precision of Defects Based on Reports Duplication in Crowdsourced Testing

KAISHUN WU, SONG HUANG, YAQING SHI, JING ZHU, AND SHIQI TANG

1 Command and Control Engineering College, Army Engineering University of PLA, Nanjing, Jiangsu 210000, China
2 Training Management Department, Navy Command College, Nanjing, Jiangsu 210000, China
3 Hunan Institute of Technology, Hengyang, Hunan 421002, China

Corresponding author: Song Huang (huangsong@aeu.edu.cn)

This work was supported by the National Key Research and Development Program of China under Grant 2018YFB1403400.

ABSTRACT When analyzing the defects of crowdsourced testing, the testing reports need to be preprocessed, including removing duplicate and false positives. At present, most crowdsourced testing research focuses on duplication of the reports, which has achieved high precision. However, studies on reducing false positives of defects have rarely been conducted. Starting from the duplication of defects in the reports, this paper discusses the relationship between the duplication and the precision of defects and proposes an estimation approach based on the defect distribution in historical crowdsourced testing projects. Experiments have shown that our approach can provide a priori knowledge of defects and exhibits good stability. We applied this approach to the defect population estimation of crowdsourced testing. With better experimental results, our improved model is more accurate than the original model. We attempt to apply it to the estimation of the defect population of crowdsourced testing, which is more accurate than the original model.

INDEX TERMS Precision and duplication of defects, crowdsourced testing, a priori distribution of defects precision.

I. INTRODUCTION

Software testing, which is recognized as a time-consuming and expensive process, can also be conducted using crowdsourcing [1]. To support the quality of crowdsourced software testing, a testing report is a vehicle for recording and tracking defects. In crowdsourced testing, workers submit corresponding testing reports to a crowdsourced testing platform, which pays remuneration according to the workload of crowdsourced testing workers. This method not only encourages crowdsourced testing workers, but also attracts more workers to participate in crowdsourced testing activities. An important basis for calculating the payment of crowdsourced testing workers is the number of true defects submitted by crowdsourced testing workers. Under this mechanism, numerous crowdsourced testing workers test the same project synchronously in parallel, while the submitted testing reports inevitably have to deal with two problems: one is duplicate, in which the same defect of the unit under test is found by multiple crowdsourced testing workers, which results in many duplicate testing reports; another is that due to their own lack of ability or other reasons, the crowdsourced testing workers submitted the false defect to the crowdsourced testing platform to obtain more rewards.

Even in traditional software testing, the problems of duplicates and false positives have received much attention. According to the defect report library of large commercial software, such as Eclipse and Firefox, 20%-40% of test reports are marked as duplicates [2]. In many cases, the judgment of the truth or falsity of defects is highly dependent on practitioners or developers in the relevant field of the unit under test [3], and sometimes even requires the help of professional testing tools [4]. Anvik et al. [5] found that less than 50% of the reports submitted in the first round of traditional software testing contained true defects.

In the study of these two problems of traditional software testing on reports, there has been some research on the problem of false positives, which are usually scoped
to white box testing. Based on the analysis of a series of false-positive elimination techniques, Zhao [6] proposed a defect detection method that combines the strength of forward dataflow analysis and backward constraint query techniques. Anant [7] proposed a transformer-based learning approach to identify false-positive bug warnings. In addition, Pistoia [8] provided a system for eliminating false-positive reports resulting from the static analysis of computer software, which includes a modeler, static analyzer, precondition generator, and precondition checker.

The present research focuses more on the problem of testing reports in duplicate. Runeson [9] took the lead in studying the duplicate detection of testing reports and used the Sony Ericsson Mobile Communications’ problem report library as its experimental data set. After quantifying the text of the testing reports, the similarity between the reports was calculated and used to detect duplicates with an accuracy rate of approximately 30%. Based on the research of Runner, Wang [10] combined the execution information of the software and defined two report similarities: natural language similarity and execution information similarity. The experimental results achieved good results in the detection rate and accuracy rate, which were 93% and 67%, respectively. The disadvantage of this method is that not all software test execution information is available. Hiew [11] proposed a defect duplicate detection method based on incremental clustering of natural language information. Kaushik [12] compared the performance of vector space models and topic-based models in detecting duplicate reports. The three topic-based models selected were LSI [13], LDA [14], and random projections [15]. The experimental results showed that the vector space model performed better. Sun [16] proposed an SVM-based discrimination model to determine the likelihood of duplication between two testing reports. Sun and Khoo [17] proposed the BM25Ext (extended BM25) method to calculate the similarity between testing reports in view of the fact that BM25 is not suitable for repeating word and long text queries.

In the field of crowdsourced testing, existing research has primarily focused on the problem of defect duplication. Wang [18] proposed a cluster-based classification method and used the integration method to build a classifier based on the most similar clusters, which improved the accuracy of existing baselines by 17% to 63% and a recall rate of 15% to 61%. Later, Wang considered that cross-domain data differences under crowdsourced testing would have an impact on cross-domain testing report classification models [19] without using historical data to train classifiers, and proposed a cross-domain classification model using stack denoising autoencoders to automatically learn advanced features from the original text and classify testing reports. Feng [20] proposed a prioritization technique based on multi-objective optimization using screenshots and text for checking crowdsourced testing reports. Liu and Zhang [21] proposed a fully automated technique to generate descriptive words for screenshots and build language models using testing reports by professional testers. Huang [22] proposed an automatic processing method based on the vector space model and used it with the similarity measurement method to detect the correctness of testing reports. Chen [23] proposed an automated detection method for testing reporting based on the BM25 algorithm that can correctly judge most testing reports to effectively improve the efficiency of identifying false reports and duplicates.

II. MOTIVATION

The problems of duplicate and false positives in crowdsourced testing reports are critical at the current stage. Existing research is mostly concerned with the duplication problem and has achieved more significant results. For the most advanced model [18], an average precision of 89% and recall of 97% were achieved. However, in the present research on automatically removing false positive defects of crowdsourced testing, only Huang’s study [22] has been involved, and the results are not ideal, with 60% accuracy and 43% recall on average.

In fact, false positive of defects in the crowdsourced testing is a key problem. As FIGURE 1, we suppose that a bug in the software under test represents a defect, and there are three true defects: Defect 1, Defect 2 and Defect 3 in the figure. In addition, Defect 4 and Defect 5 are normal modules of the software under test, which was falsely reported as a defect by crowdsourced testing workers. We have to remove these false defects in various ways when integrating crowdsourced testing reports.

When analyzing the false positive of crowdsourced testing defects, the correlation between it and the duplicate of reports was found. TABLE 1 is duplicate defect statistics for a unit under test after deduplication for final crowdsourced testing report, where each column corresponds to the defect, each row corresponds to all the test reports submitted by the crowdsourced testing worker to this unit under test. When the value of column j in row i is 1, it means that the ith crowdsourced testing worker submitted /th defect; otherwise, it is 0. Columns with a gray background correspond to the defects of false positive.

The differences between the true and false defects in TABLE 1, it can be found that the column 3, 10 and 17 corresponding to false defects, only 1, 2 and 1 of the
TABLE 1. Duplicate defect statistics for a unit under test (including false positive).

| ID| Defect |
|---|---|
| 1 | 0 1 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 |
| 2 | 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 |
| 3 | 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 0 |
| 4 | 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0 1 0 |
| 5 | 0 1 0 1 0 0 1 0 0 0 0 0 0 0 0 1 0 |
| 6 | 0 1 0 0 0 0 0 0 1 0 0 0 0 1 1 0 0 |
| 7 | 0 1 0 1 0 0 1 1 0 1 0 0 0 0 1 0 1 0 |
| 8 | 0 1 0 1 0 0 1 0 1 0 0 1 0 1 1 0 0 |
| 9 | 0 1 0 1 0 0 1 0 0 1 0 0 0 1 0 0 1 0 |
| 10 | 0 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 |
| 11 | 0 0 0 0 0 0 1 0 1 0 0 1 1 0 0 1 0 |
| 12 | 0 0 0 0 0 1 0 1 0 0 0 0 0 0 1 0 |
| 13 | 0 1 0 0 0 0 0 1 1 1 0 1 1 1 0 0 0 |
| 14 | 0 0 0 0 0 1 0 1 0 1 1 0 0 0 1 0 0 |
| 15 | 0 1 0 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 |
| 16 | 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 |
| 17 | 0 1 0 0 0 0 0 0 1 1 0 1 1 0 0 0 |
| 18 | 0 0 0 0 0 1 0 1 0 1 0 0 1 0 1 0 |
| 19 | 0 1 0 1 1 0 0 1 1 0 0 0 1 0 0 0 |
| 20 | 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 |

SUM: 1 1 3 1 5 6 3 6 7 16 2 4 5 8 5 5 6 1

20 crowdsourced testing workers submitted the defects, respectively. For the true defects, although there is a defect submitted by only one person, the average number of submitted true defects is 6.42, which is far above the average number of false defects. Thus, we can assume that for one defect, the greater the number of submissions by crowdsourced testing workers, the higher the probability that this defect will be true.

Currently, there is no research that considers false positive problems based on the results of removing duplicate reports of crowdsourced testing, which is the entry point of this study. Therefore, we collected the actual defects of duplicate and false positives on the crowdsourced test platforms of CoForTest [24] and MoocTest [25]. We filtered these testing reports and removed those without defects. In addition, we cut the testing report according to the number of defects, ensuring that one test report has only one defect; however, the defect may be true or false. The statistics of the reports are shown in TABLE 2.

TABLE 2. Statistics for crowdsourced testing reports to defect analysis.

| Platform | Projects | Total defects | False defects | True defects |
|---|---|---|---|---|
| MoocTest | 16 | 437 | 103 | 334 |
| CoForTest | 37 | 1779 | 735 | 1044 |

In this case, we divided the defects into eight categories according to the number of submissions from one to eight and the number of last category is the submissions of eight or more. For crowdsourced testing projects, the precision was calculated using Equation (1) as follows:

\[ P_i = \frac{TP_i}{TP_i + FP_i} \]  

(1)

where \( TP_i \) is the number of true defects with a submission number of \( i \), \( FP_i \) is the number of false defects that had a submission number of \( i \), and \( P_i \) is the precision with a submission number of \( i \) in this project.

Based on the projects, the relationship between statistical precision and the number of defect submissions, with precision as the ordinate, the submission number of defects as the abscissa, box plot, and average trend line, can be drawn as shown in FIGURE 2.

![FIGURE 2. Precision vs. duplication.](image)

It is clear that for an individual defect, the greater the submission number, the higher the likelihood that the defect would be true. Among the defects with a submission number of 1, the ratio of true and false defects is close to 1:1 with an average precision of 49.66%; among the defects with a submission number of 2, the ratio of true and false defects is close to 3:1 with an average precision of 75.97%. As the submission number increases, so does the average precision, eventually converging to approximately 96% (according to the different platforms and units under test).

### III. APPROACH

#### A. PROBABILITY MODEL FOR CROWDSOURCED TESTING TASKS

We attempted to represent and explain this phenomenon using a probability model. Considering an extreme situation, we designed the crowdsourced testing task to be the smallest, in which there would be at most only one true defect. At present, we call this the minimal crowdsourced testing task (such crowdsourced testing tasks are generally not deliberately designed in actual crowdsourced testing projects).

The minimum crowdsourced testing task \( T \) is set to have its confusion matrix, as shown in TABLE 3.

TABLE 3. Defect confusion matrix.

| Confusion Matrix | Positive Report | Negative Report |
|---|---|---|
| Defect | TP(True Positive) | FP(False Positive) |
| Positive | FN(False Negative) | TN(True Negative) |
In the minimum crowdsourced testing task $T$, the four cases are listed in Table 3 are represented as probability:

$$ pTP + pFP + pTN + pFN = 1 \quad (2) $$

Based on the minimum crowdsourced testing task, the minimum test report $Re$ is introduced, that is, if a crowdsourced testing task is submitted by a crowdsourced testing worker, who must submit only one report, this report can only contain at most one defect. We assume that all defect probabilities for $T$ are $D = \{d_1, d_2, \ldots, d_N\}$. If $T$ does not contain defects, the report submitted by crowdsourced testing workers is divided into the following two cases:

1. The worker’s report showed that there are no defects after the task $T$ was executed, corresponds to confusion matrix $TN$;
2. The worker reported defect $d_k$ after the task $T$ was executed, corresponds to confusion matrix $FP$.

There were no cases of $TP$ and $FN$ in the confusion matrix, where $pTP = pFN = 0$.

If the minimum crowdsourced testing task $T$ contains a true defect of $d_k$, the following three cases must be considered:

3. The worker reported true defect $d_k$ after the task $T$ was executed, corresponds to confusion matrix $TP$;
4. The worker did not report defects after the task $T$ was executed, corresponds to confusion matrix $FN$;
5. The worker reported false defect $d_s$, $s \neq k$ after the task $T$ was executed, corresponds to confusion matrix $FP$.

There was no case of $FN$ in the confusion matrix, where $pFN = 0$.

In the cases of 1) and 2), consider two crowdsourced testing workers, $n$ and $m$ both submitted reports. In the report submitted by crowdsourced testing worker $n$, all possibilities corresponding to crowdsourced testing defect set $D$ are

$$ p_n[d_i \in D] = \{p_{n_1}, p_{n_2}, \ldots, p_{n_N}\}, \text{in which} \sum_{i=1}^{N} p_{n_i} = pFN. $$

For a uniform representation, we set the case of no defects for the report to be $d_0$ corresponding to 1). We make $P[d_0] = p_{n_0} = p_{m_0}$, and expend $D$ to $D^* = D \cup \{d_0\}$.

At present, the probability distribution of the report submitted by tester $n$ can be expressed as:

$$ p_n[d_i \in D^*] = \{p_{n_0}, p_{n_1}, p_{n_2}, \ldots, p_{n_N}\} \quad (3) $$

where $\sum_{i=0}^{N} p_{n_i} = 1$. Similar to crowdsourced testing worker $m$, the probability distribution of $D^*$ is:

$$ p_m[d_i \in D^*] = \{p_{m_0}, p_{m_1}, p_{m_2}, \ldots, p_{m_N}\} \quad (4) $$

where $\sum_{i=0}^{N} p_{m_i} = 1$.

In the case of 3), 4), and 5), a hypothesis is added to the above, which makes the 1st defect in $D$ correspond to the true defect, which let $d_1 = d_v$. Then, for defects submitted by two crowdsourced testing workers, $n$ and $m$, we can still use Equation(3) and Equation(4) to represent their probability distribution.

In practice, scenarios are often superimposed from multiple defect probability distributions of the minimum crowdsourced testing task of a single submission, such as the dimension of crowdsourced testing workers, multiple defects from a single task, and numerous crowdsourced testing tasks.

For example, we assume that crowdsourced testing workers $n$ and $m$ do not communicate when testing task $T$ and submit reports; that is, $Pn$ and $Pm$ are independent, and the joint probability of $Pn$ and $Pm$ is:

$$ P(n_i, m_j) = P(n_i)P(m_j) = p_{n_i}p_{m_j} \quad (5) $$

**B. SAMPLING TO ESTIMATED PRECISION**

Based on the probability model of crowdsourced testing tasks, we estimate the precision of the reports for ongoing crowdsourced testing projects based on all project records. According to historical experience, the more similar the main participants in a crowdsourced testing project, the more similar is the precision distribution of the corresponding defects. Specifically, the factors that affect the final quality of crowdsourced testing are as follows:

1. Participating stakeholders as: crowdsourced testing workers $W$, crowdsourced testing platform $C$, unit under test $U$, testing requirement $Q$;
2. Relating strategies as: crowdsourced testing project duration $Du$, reward amount $M$ and incentive $I$, assignment of crowdsourced testing tasks $A$;
3. Resulting data as: crowdsourced testing tasks $T$, reports submitted by workers $Re$, crowdsourced testing final report $Fr$;
4. Other factors as $O$.

We use all factors to describe this crowdsourced testing project, denoted as $E$, such that:

$$ CP = E(U, Q, W, C, I, Du, M, A, T, Re, Fr, O) \quad (6) $$

For the defect distribution in the testing reports, we use $P(X; Re, D)$ to represent. The relationship between the distribution of the defect set $Re$ in the submitted testing reports and crowdsourced testing project $CP$ can be defined as:

$$ RCP_{-(D, Re)} = \{\langle CP_i, P(X; Re, D) \rangle \}, i = 1, 2, 3 \ldots \quad (7) $$

which is the relationship that exists in the probability space $\mathcal{F}$ of crowdsourced testing, and can be used to evaluate the crowdsourced testing quality.

However, in reality, there is no way to exhaust all defects in $Re$, we describe defects in terms of the distribution of repetitions of defects. First, we define the function $f_{count}$ to calculate the number of report repetitions according to the defect in $Re$ as:

$$ f_{count}(i; D, Re) = \{\text{The number of reports repetitions is } i \text{ according to the defect in } Re, \text{ accumulating all these defects.} \} \quad (8) $$

Here, we ignore the corresponding cases for reports without defects in 1) and 4) in Section III-A. Simultaneously, to facilitate the distinction between true and false defects, we use $Re_t$.
to represent the true defect set, with \( Re_f \) for the false. And it is clear that:

\[
Re = Re_t \cup Re_f
\]  
(9)

and

\[
Re_t \cap Re_f = \emptyset
\]  
(10)

Then, we calculate the defect–duplication report distribution as:

\[
P(X; Re, D) = P(X = x; Re, D) = \frac{f_{\text{count}}(x; D, Re)}{\sum_{i=1}^{n} f_{\text{count}}(i; D, Re)}
\]  
(11)

Of course, the defect precision of the duplicate distribution that we are more concerned with can also be calculated as:

\[
P(X|Re_t; Re, D) = \frac{f_{\text{count}}(x; D, Re_t)}{\sum_{i=1}^{n} f_{\text{count}}(i; D, Re)}
\]  
(12)

Of course, the defect precision of duplicate distribution is also related to the defect distribution of the testing reports, and we express this relationship as:

\[
R_{(D,Re)-Re_t} = \{(P_i(X|Re_t, D), P_i(X|Re_t, D))\} \quad \text{(13)}
\]

Thus, according to Equation(7) and Equation(13), we can construct a new relationship \( R_{CP-(D,Re_t)} \), notation as:

\[
R_{CP-(D,Re_t)} = \{(R_{CP-(D,Re)}, R_{(D,Re)-Re_t})\}
\]  
(14)

When considering the actual situation of crowdsourced testing when executed, we can reduce the partial variables in Equation(6). Considering that the units under test and testing requirements are difficult to calculate directly as variables, we instead use them as the size of the unit under test \( S \) and testing difficulty \( L \). Next, we select crowdsourced testing workers \( W \), crowdsourced testing project duration \( Du \), reward amount \( M \), and reports submitted by workers \( Re \) to represent the crowdsourced testing project because these variables facilitate acquisition and measurement in crowdsourced testing. From this, the relationship between the crowdsourced testing project attributes and defect precision of the duplicate distribution can be constructed as follows:

\[
R_{CP_a-(D,Re_t)} = \{(E(S, L, W, Du, Re), P(X|Re_t; Re, D))\}
\]  
(15)

Based on this relationship, we can filter historical crowdsourced testing projects that are similar to the given crowdsourced testing project to estimate the defect precision of duplicate distributions for this project. Set up an ongoing crowdsourced testing project \( CP_n = \{S_n, L_n, W_n, Du_n, [Re_n], . . . . \} \), and completed historical crowdsourced testing projects set \( CP = \{CP_0, CP_1, . . . . , CP_{n-1}\} \), that \( CP_i = \{S_i, L_i, W_i, Du_i, [Re_i], . . . . \} \), \( i = 0, 1, 2, . . . , n - 1 \).

For the convenience of the following calculations, we use a set of scalars to represent the crowdsourced testing projects:

\[
CP = \{cp_1, cp_2, cp_3, . . . . , cp_m\}, \text{ where } m \text{ is the number of } S \text{ filtered for the crowdsourced testing projects.}
\]

At the same time, we express the reports corresponding to the crowdsourced testing project as a separate indicator, where \( Re_t = \{(w_{i1}, d_{i1}), (w_{i2}, d_{i2}), (w_{i3}, d_{i3}), . . . . , (w_{ik}, d_{ik})\} \), where \( k \) is the number of reports and \( w_{ij} \) \( \in \) \( W \) represents the crowdsourced testing worker who submitted the \( j \)th report for the crowdsourced testing project \( CP_n, d_{ik} \in D \) represents the defect corresponding to the \( j \)th submitted report in \( CP \). The handling of the reports here is the same as III-A; that is, a report must and can only contain one defect(reports without defects have been rejected). Next, we assume that the reports in the historical projects have completed the determination of duplicate and false positives of defects. Setting the filter range to \( k \) and the maximum number of duplicate reports to \( x \), we combine the historical crowdsourced testing project set \( CP \) and ongoing crowdsourced testing project \( CP_n \) with their corresponding reports \( Re \) and \( Re_n \), we can estimate the defect precision of \( CP_n \) using the Algorithm 1.

**Algorithm 1** Estimate the Precision of Projects That Are Being Crowdsourced Tested

**Input:** \( (CP, Re), (CP_n, Re_n), k, x, Re_t \)

**Output:** \( \hat{P}(X|Re_t; Re, D) \)

1: for each \( CP_i \in CP \cup \{CP_n\} \) do
2: for each \( cp_{i,j} \in CP_i \) do
3: \( cp_{i,j} = \frac{cp_{i,j} - \min_{i=1}^{n} cp_{i,j}}{\max_{i=1}^{n} cp_{i,j} - \min_{i=1}^{n} cp_{i,j}} \)
4: end for
5: \( CP' = \{cp'_{1,1}, cp'_{1,2}, . . . . , cp'_{n,m}\} \)
6: end for
7: for each \( CP_i \in CP' \) do
8: \( dist_i = \cos(cp_i', CP_n') \)
9: end for
10: \( Dist = \min_{i=1}^{n} \{dist_i\} \)
11: for \( Re_e = \emptyset; (k > 0) \) and \( (Dist \neq \emptyset); k = k - 1 \) do
12: \( indexmin = indexof(\min Dist) \)
13: \( Dist = Dist - \min Dist \)
14: \( Re_e = Re_e \cup \{Re_{indexmin}\} \)
15: end for
16: end for
17: for each \( Re_t \in Re_e \) do
18: \( Re_t = \text{splice}(0, [Re_{index}]) / / f l e t o r r e p o r t i s i n t e r p e t a t e d t o b e a s l o n g o r s m a l l a s t h e r e p o r t t o b e e s t i m a t e d \)
19: end for
20: for \( j = 1; j < x; j = j + 1 \) do
21: \( \hat{P}(X = j|Re_t, Re_n, D) = \frac{f_{\text{count}}(j; Re_t, D)}{\sum_{j=1}^{\infty} f_{\text{count}}(j; Re_t, D)} \)
22: end for
23: \( \hat{P}(X \geq x|Re_t, Re_n, D) = \frac{\sum_{j=x}^{\infty} f_{\text{count}}(j; Re_t, D)}{\sum_{j=1}^{\infty} f_{\text{count}}(j; Re_t, D)} \)
The main idea of this algorithm is to normalize the indicators for all crowdsourced testing projects, use the cosine to calculate the distance between \( CP_n \) and the historical projects to find the \( k \) nearest projects, crop the filtered projects to keep only the report sequence equal to or shorter than the length of \( CP_n \), and then calculate the average distribution of the clipped historical reports, which is the estimate of the defect precision of duplicate distribution for \( CP_n \).

Here, the value of \( K \) can be determined using KL divergence. The relationship between the crowdsourced testing project attribute and defect precision of the duplicate distribution in Equation(15). We choose the value of \( K \) by minimizing KL divergence, which can be calculated as:

\[
KL(CP_n||CP) = \sum_{i=1}^{m} cp_n \log \frac{cp_n}{cp_i} \tag{16}
\]

The corresponding \( K \) value was chosen by minimizing \( KL(CP_n||CP) \). Of course, in the initial stage of the crowdsourced testing platform, when the accumulated historical projects are insufficient, all projects on the platform can be directly used for estimation.

IV. EXPERIMENT

The experiments in our study are mainly divided into two aspects: one is to verify the precision of the method in III-B, and the other is to verify the effectiveness of the precision distribution of defect duplication applied in crowdsourced testing.

We measure the precision of the estimation based on the magnitude of the relative error (MRE), which is the most commonly used measure for precision. It measures the relative error ratio between the actual and estimated values, expressed as follows:

\[
\text{MRE} = \frac{\text{predicted value} - \text{actual value}}{\text{actual value}} \tag{17}
\]

Note that, to more clearly indicate the positive and negative values of the error, there is no absolute value for the MRE.

A. PRECISION EXPERIMENT

To verify the precision of our approach, we extracted relevant information affecting the quality of the test for all crowdsourced testing projects in TABLE 2. Subsequently, we selected 11 of the 53 crowdsourced testing projects as testing sets and considered the truth value of the defects as unknown. The precision of defects in the test set was estimated using the precision distribution of defect duplication of another 42 projects, with the related information listed in TABLE 4.

| TABLE 4. Statistics of data set for precision experiment. |
|-----------------|-----------------|-----------------|-----------------|
| Data set        | Projects | Total defects | False defects | True defects |
| Training set    | 42       | 3145          | 595           | 2589         |
| Test set        | 11       | 789           | 112           | 513          |

Then, we take the maximum number of repetitions \( X \leq 8 \) with the filter range of \( k = 23 \), the MRE of precision distribute of defects duplication could be calculated as:

\[
\text{MRE}(x, Re_n) = \frac{\hat{P}(X = x|Re_n; D) - P(X = x|Re_n; D)}{P(X = x|Re_n; D)} \tag{18}
\]

From this, the scatter plot of the results for the 11 projects that we used to estimate can be drawn as FIGURE 3.

FIGURE 3. Scatter plot of estimated precision MRE of defects for precision experiment.

It can be observed that the mean of the MRE is uniformly distributed around the line of \( y = 0 \). More details are provided in TABLE 5 showed the mean and median of MRE according to the defect duplicate. The average MRE is 0.0279 < 0.05, which indicates good performance. The average MSE was 0.1065 < 0.68, which means that our data had a low degree of dispersion. The distribution of the MRE, that is, our prediction results, is also relatively stable.

| TABLE 5. Average MRE and MSE of estimated results. |
|----------------|----------------|----------------|----------------|
| Duplicate | 1   | 2   | 3   | 4   | 5   |
| MRE-avg    | -0.0622 | 0.0496 | 0.0147 | 0.0289 | -0.0119 |
| MRE-avg-abs | 0.1604 | 0.1802 | 0.1781 | 0.1153 | 0.056 |
| MSE        | 0.1567 | 0.1378 | 0.1533 | 0.0736 | 0.0479 |

From TABLE 5 and FIGURE 3, we can see that in our method has a relatively large MSE when the number of duplicate defects is less than 4. However, when the number of duplicate defects reached four or more, the MSE gradually decreased to the low level. Using our approach to estimate the precision of duplicate defects at this time, the performance was better.

However, from the experimental results of these projects in our dataset, the MRE value obtained the minimum absolute
value and minimum variance when the number of defect duplicates reached five, and gradually increased after the number of repeats was greater than five. This phenomenon is caused by certain errors.

**TABLE 6.** Statistics on the reasons of defects false positives with a number of defects duplicate greater than 5.

| Reason                        | Number | Proportion |
|-------------------------------|--------|------------|
| Requirements conflict         | 22     | 30.14%     |
| Suggestions for improvement, not defects | 31     | 42.47%     |
| Unclear or misleading tasks   | 17     | 23.29%     |
| Plagiarism                    | 9      | 12.33%     |
| Unknown                       | 8      | 10.96%     |
| **Sum**                       | **73** | -          |

By selecting the completed crowdsourced testing projects, we counted a total of 73 false positive defects with more than 5, and the reasons for false positives are summarized in TABLE 6. Among them, approximately 30.14% of the false positive defects are caused by the conflict between the test requirements and the software specifications of the unit under test. Approximately 42.47% of false positive defects are due to the content submitted by test workers, which suggests improvement instead of defects. The unclear or misleading descriptions of crowdsourced testing tasks at the design stage led to approximately 23.29% false-positive defects. In addition, plagiarism from crowdsourced testing workers also resulted in a small number of false positive defects, around 12.33%.

**B. EFFECTIVENESS EXPERIMENT**

The problem of duplication of reports has been solved automatically in existing research, with high precision. These methods have already been used on crowdsourced testing platforms such as [24] and [25]. It is a clever way to use report duplicates to estimate the precision of defects to be used in other aspects of crowdsourced testing. We illustrate its advantages through a crowdsourced testing application scenario to estimate the defect population.

Defect duplication is widely used in crowdsourced testing. Junjie provided the iSENCE method [26] to predict the total number of defects and test costs of a crowdsourced testing project using the incremental extraction of test reports. Yao [27] used the CRC model to predict the total number of defects in a crowdsourced test scenario and evaluated the completion of a crowdsourced testing task.

Both the studies of Junjie and Yi involved a famous method, the capture-recapture method [28](CRC method), which is very easy to apply in crowdsourced testing. However, there is a very critical assumption in the CRC method that is ignored, that the defects are all true in extraction. When this assumption does not hold, it can have a significant impact on the sample population estimation. Taking crowdsourced testing as an example, when the total sample size is estimated, if the true or false nature of the defects is unknown, the estimated population of the samples will be higher than the actual total number of defects. Of course, the real-time manual determination of the submitted report in the crowdsourced testing process can solve this problem, but it will increase the cost significantly.

Using our method, it is possible to estimate the precision of defects when determining their duplicates, thereby improving the precision of the estimation of the defect population to a certain extent.

The dataset used in this experiment is shown in TABLE 1, that we would estimate the population of their true defects. We selected the M0 [29] and Mth [30] models from the most commonly used CRC methods and used them to estimate the population of true defects at each split point distribution following 10% of the defect arrival sequence as the split point.

Then, we corrected the variables of the M0 and Mth models of the crowdsourced testing project. Using Algorithm.1 to calculate the defects precision as the correction coefficient, new models of M0 and Mth are obtained on this basis.
false defects on the datasets by removing the false positives. However, after using our method to correct the original models, the new models converged to approximately $y = 0$.

C. EXPERIMENT SUMMARY

Based on the observation that the defect duplicates are positively correlated with the defect precision, we support a defect precision distribution estimation method based on the duplication of historical crowdsourced testing project data, which can have an a priori estimate of defect precision during crowdsourced testing. Two experiments were designed to verify the precision and effectiveness of the proposed method.

In the effectiveness experiment, we collected data on crowdsourced testing projects from the CoForTest and MooctTest platforms, in which 11 of the 53 projects were selected as the test set, where the truth or falsity of defects was considered unknown. The remaining 42 projects are used as the training set to estimate, so as to obtain the defect precision distribution based on the defects duplicate. We compared the experimental results with the ground truth of these projects using the MRE to evaluate the precision of the method. The average MRE in this experiment was 0.0279, and the mean squared error was 0.1065, indicating that the experimental precision and stability were good.

In the effectiveness experiment, we selected the application scenario for the crowdsourced testing completion evaluation. Combined with the traditional CRC model, our method was applied to estimate defect populations. For crowdsourced testing reports without manually removing false positives, our model significantly improves the estimation precision of the defect population. This experiment shows the positive effect of our method in creating false positive defects, the application scenarios for crowdsourced testing completion evaluation are promoted, and the technology of crowdsourced testing completion evaluation can be used for crowdsourced testing reports without manually removing false positives.

V. CONCLUSION

Our method estimates precision by the duplication of defects, which can greatly improve the efficiency of defect analysis in the crowd testing process, and can also be applied as a priori knowledge to other techniques of crowdsourced testing. In the process of estimating the defect population, our approach is integrated with the traditional CRC method and applied to the data of crowdsourced testing projects without reducing false positives, which can significantly improve the estimation precision of the true defect population. Subsequently, our method can be combined with the prioritization of crowdsourced testing reports and quality evaluations in crowdsourced testing. Thus, we explored the possibility of improving the quality and reducing the cost of crowdsourced testing.

REFERENCES

[1] R. Gao, Y. Wang, Y. Feng, Z. Chen, and W. E. Wong, “Successes, challenges, and rethinking—An industrial investigation on crowdsourced mobile application testing,” Empirical Softw. Eng., vol. 24, no. 2, pp. 537–561, Apr. 2019.
[2] J. Anvik, L. Hiew, and G. Murphy, “Coping with an open bug repository,” in Proc. OOPS LA Workshop, Eclipse Technol. eXchange, Jan. 2005, pp. 35–39.
[3] D. Li, W. E. Wong, S. Pan, L.-S. Koh, S. Li, and M. Chau, “Automatic test case generation using many-objective search and principal component analysis,” IEEE Access, vol. 10, pp. 85518–85529, 2022.
[4] C. Hsin-Yu, H. Chin-Yu, and F. Chih-Chiang, “Applying slicing-based testability transformation to improve test data generation with symbolic execution,” Int. J. Performability Eng., vol. 17, no. 7, p. 589, 2021.
[5] J. Anvik, L. Hiew, and G. C. Murphy, “Who should fix this bug?,” in Proc. 28th Int. Conf. Softw. Eng., May 2006, pp. 361–370.
[6] Y. Zhao, Y. Gong, Q. Wang, A. Zhou, and H. Zhou, “False positive elimination in static defect detection,” JSunshine Sunjie Yu Zhishan Comput. Devolop., vol. 49, pp. 1822–1831, Jul. 2012.
[7] A. Kharkar, R. Z. Moghaddam, M. Jin, X. Liu, X. Shi, C. Clement, and N. Sandu, “Learning to reduce false positives in analytic bug detectors,” in Proc. 44th Int. Conf. Softw. Eng., May 2022, pp. 1307–1316.
[8] T. Pistoia and O. Marco, “Eliminating false-positive reports resulting from static analysis of computer software,” U.S. Patent 8,745,578 B2, Jun. 3, 2013.
[9] F. Runeson, M. Alexandersson, and O. Nyholm, “Detection of duplicate defect reports using natural language processing,” in Proc. 29th Int. Conf. Softw. Eng. (ICSE), May 2007, pp. 499–510.
[10] 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 Proc. 30th Int. Conf. Softw. Eng., Jan. 2008, pp. 461–470.
[11] L. Hiew, “Assisted detection of duplicate bug reports,” Univ. British Columbia, Vancouver, Canada, 2006.
[12] N. Kaushik and L. Tahvildari, “A comparative study of the performance of IR models on duplicate bug detection,” in Proc. 16th IEEI Conf. Softw. Maintenance Reeng., Mar. 2012, pp. 159–168.
[13] S. Deere, S. T. Dumas, G. W. Furnas, T. K. Landauer, and R. Harshman, “Indexing by latent semantic analysis,” J. Amer. Soc. Inf. Sci., vol. 41, no. 6, pp. 391–407, 1990.
[14] D. M. Blei, A. Ng, and M. I. Jordan, “Latent Dirichlet allocation,” J. Mach. Learn. Res., vol. 3, pp. 993–1022, Jul. 2003.
[15] P. Kaper, J. Kristoferson, and A. Holst, “Random indexing of text samples for latent semantic analysis,” in Proc. Annu. Meeting Cogn. Sci. Soc., vol. 1036, Aug. 2000, pp. 1–2.
[16] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Koo, “A discriminative model approach for accurate duplicate bug report retrieval,” in Proc. ICSE, vol. 1, 2010, pp. 45–54.
[17] C. Sun, D. Lo, S.-C. Koo, and J. Jiang, “Towards more accurate retrieval of duplicate bug reports,” in Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE), Nov. 2011, pp. 253–262.
[18] J. Wang, Q. Cui, Q. Wang, and S. Wang, “Towards effectively test report classification to assist crowdsourced testing,” in Proc. 10th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas., Sep. 2016, pp. 1–10.
[19] J. Wang, S. Wang, Q. Cui, and Q. Wang, “Local-based active classification of test report to assist crowdsourced testing,” in Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng., Aug. 2016, pp. 190–201.
[20] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, “Multi-objective test report prioritization using image understanding,” in Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng., Aug. 2016, pp. 202–213.
[21] D. Liu, X. Zhang, Y. Feng, and J. A. Jones, “Generating descriptions for screenshots to assist crowdsourced testing,” in Proc. IEEE 25th Int. Conf. Softw. Anal. Evol. Reengineering (SANER), Mar. 2018, pp. 492–496.
[22] S. Huang, L. Chen, Z. Hui, J. Liu, S. Yang, and Q. Chen, “A method of bug report quality detection based on vector space model,” in Proc. IEEE 19th Int. Conf. Softw. Qual. Rel. Secur. Companion (QRS-C), Jul. 2019, pp. 510–511.
[23] L. Chen, S. Huang, J. Sun, and Z. Hui, “Bug report quality detection based on the bm25 algorithm,” J. Tsinghua Univ. Sci. Technol., vol. 60, no. 10, pp. 1–8, 2020.
[24] (Jun. 2022). Cofo تست. [Online]. Available: http://www.cofo تست/.
[25] Mooctест. Accessed: Jun. 2022. [Online]. Available: http://www.mooctest.net
[26] J. Wang, Y. Yang, R. Krishna, T. Mengas, and Q. Wang, “ISENSE: Completion-aware crowdtesting management,” in Proc. IEEE/ACM 41st Int. Conf. Softw. Eng. (ICSE), May 2019, pp. 912–923.
[27] Y. Yao and Y. Liu, “Application and improvement of capture-recapture model for crowdsourced testing,” J. Phys. Conf. Ser., vol. 1738, no. 1, Jan. 2021, Art. no. 012125.
K. Wu et al.: Estimate the Precision of Defects Based on Reports Duplication in Crowdsourced Testing

[28] J. Darroch, “The multiple-recapture census: I. Estimation of a closed population,” *Biometrika*, vol. 45, pp. 343–359, Dec. 1958.

[29] G. Liu, G. Rong, H. Zhang, and Q. Shan, “The adoption of capture-recapture in software engineering: A systematic literature review,” in *Proc. 19th Int. Conf. Eval. Assessment Softw. Eng.*, Apr. 2015, pp. 1–13.

[30] S.-M. Lee, “Estimating population size for capture-recapture data when capture probabilities vary by time, behavior and individual animal,” *Commun. Statist. Simul. Comput.*, vol. 25, no. 2, pp. 431–457, Jan. 1996.

**KAISHUN WU** is currently pursuing the Ph.D. degree in software engineering with Army Engineering University. His research interests include software testing, crowdsourced testing, software evaluation, and assessment.

**SONG HUANG** received the B.S., M.S., and Ph.D. degrees from the PLA University of Science and Technology. He is currently a Professor of software engineering with the Software Testing and Evaluation Center, Army Engineering University of PLA. His research interests include software testing and empirical software engineering. He is a member of the Advisory Boards of *Journal of Systems and Software* and *IEEE Transactions on Reliability*.

**YAQING SHI** received the Ph.D. degree in computer application technology from Hohai University, China, in 2015. She is currently a Professor with the Command and Control Engineering College, Army Engineering University of PLA, Nanjing, China. Her research interests include software testing, quality assurance, spatiotemporal indexing, and searching methods.

**JING ZHU** received the B.E. degree from the Jiangsu University of Science and Technology, in 2006, and the M.E. degree from the Navy Command College, in 2011. He is currently pursuing the Ph.D. degree in software engineering with the Army Engineering University of PLA. He joined the Navy Command College as a Lecturer, in 2013. His research interests include software engineering, pattern recognition, and intelligent systems.

**SHIQI TANG** received the Ph.D. degree in software engineering from the Army Engineering University of PLA, in 2022. His research interests include fault localization, defect detection, and machine learning.