2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)

iSENSE: Completion-Aware Crowdtesting Management

Junjie Wang[1,2,3], Ye Yang[4], Rahul Krishna[5], Tim Menzies[5], Qing Wang[1,2,3,*]
[1]*Laboratory for Internet Software Technologie,s* [2]*State Key Laboratory of Computer Science*
Institute of Software Chinese Academy of Sciences, Beijing, China;
[3]*University of Chinese Academy of Sciences, Beijing, China;* [*]*Corresponding author.*
[4]*School of Systems and Enterprises, Stevens Institute of Technology, Hoboken, NJ, USA;*
[5]*Department of Computer Science, North Carolina State University, Raleigh, NC, USA*
wangjunjie@iscas.ac.cn; ye.yang@stevens.edu; rkrish11@ncsu.edu; tim@menzies.us; wq@iscas.ac.cn

***Abstract*—Crowdtesting has become an effective alternative to traditional testing, especially for mobile applications. However, crowdtesting is hard to manage in nature. Given the complexity of mobile applications and unpredictability of distributed crowdtesting processes, it is difficult to estimate (a) remaining number of bugs yet to be detected or (b) required cost to find those bugs. Experience-based decisions may result in ineffective crowdtesting processes, e.g., there is an average of 32% wasteful spending in current crowdtesting practices.**

This paper aims at exploring automated decision support to effectively manage crowdtesting processes. It proposes an approach named ISENSE which applies incremental sampling technique to process crowdtesting reports arriving in chronological order, organizes them into fixed-size groups as dynamic inputs, and predicts two test completion indicators in an incremental manner. The two indicators are: 1) total number of bugs predicted with Capture-ReCapture model, and 2) required test cost for achieving certain test objectives predicted with AutoRegressive Integrated Moving Average model. The evaluation of ISENSE is conducted on 46,434 reports of 218 crowdtesting tasks from one of the largest crowdtesting platforms in China. Its effectiveness is demonstrated through two application studies for automating crowdtesting management and semi-automation of task closing trade-off analysis. The results show that ISENSE can provide managers with greater awareness of testing progress to achieve cost-effectiveness gains of crowdtesting. Specifically, a median of 100% bugs can be detected with 30% saved cost based on the automated close prediction.

***Keywords*-Crowdtesting; automated close prediction; test completion; crowdtesting management;**

I. Introduction

Crowdtesting is an emerging paradigm which can improve the cost-effectiveness of software testing and accelerate its process, especially for mobile applications [1]–[5]. It entrusts testing tasks to online crowdworkers whose diverse testing environments, background, and skill sets could significantly contribute to more reliable, cost-effective, and efficient testing results [4], [5]. Crowdtesting has been adopted by many software organizations, including but not limited to Google, Facebook, Amazon, Microsoft [6], [7]. Specifically, Google regularly deploys crowdtesting for 14 of their major product lines [6]. A latest report by Gartner Inc. predicts that crowdtesting will constitute 20% of all enterprise application testing initiatives by 2018 [8].

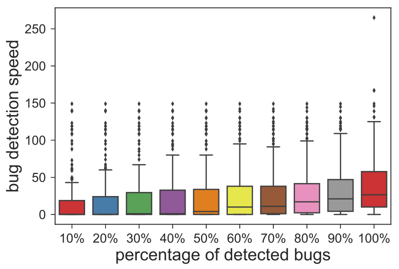
Trade-offs such as "how much testing is enough" are critical yet challenging project decisions in software engineering [9]–[12]. Insufficient testing can lead to unsatisfying software quality, while excessive testing can result in potential schedule delays and low cost-effectiveness. This is especially true for crowdtesting given the complexity of mobile applications and unpredictability of distributed crowdtesting processes.

In practice, project managers typically plan for the close of crowdtesting tasks solely based on their personal experience. For example, they usually employ duration-based or participant-based condition to close crowdtesting tasks through either a fixed period (e.g., 5 days) or a fixed number of participant (e.g., recruiting 400 crowdworkers). If either of the criteria is met first, the task will be automatically closed. However, our investigation on real-world crowdtesting data (Section II-C) reveals that there are large variations in bug arrival rate of crowdtesting tasks, and in task's duration and consumed cost for achieving the same quality level. It is very challenging for managers to come up with reasonable decisions. These experience-based decisions could result in ineffective crowdtesting process, e.g. an average of 32% wasteful spending in our experimental crowdtesting platform (Section II-C). Furthermore, crowdtesting is typically treated as a black box process and managers' decisions remain insensitive to its actual progress. This suggests the practical need and potential opportunity to improve current crowdtesting practices.

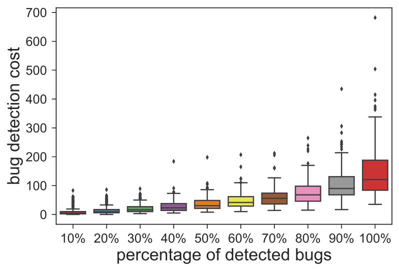
This paper aims at exploring automated decision support to raise completion awareness w.r.t. crowdtesting processes, and manage crowdtesting practices more effectively. Particularly, we leverage dynamical bug arrival data associated with crowdtesting reports, and investigate whether it is possible to determine that, at certain point of time, a task has obtained satisfactory bug detection level.

The proposed completion-aware crowdtesting management approach ISENSE[1] applies incremental sampling tech-

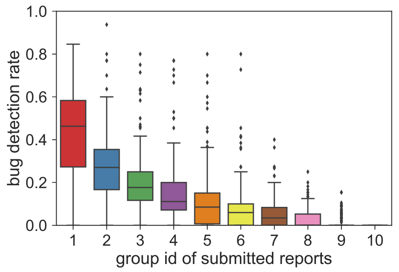
[1]ISENSE is named considering it is like a sensor in crowdtesting processes to raise the awareness of the testing progress.



(a) Bug detection speed



(b) Bug detection cost



(c) Bug detection rate

Figure 1: Observations on real-world crowdtesting data

nique to process crowdtesting reports arriving in chronological order, organizes them into fixed-size groups as dynamic inputs, and integrates Capture-ReCapture (CRC) model and Autoregressive Integrated Moving Average (ARIMA) model to raise awareness of crowdtesting progress. CRC model is widely applied to estimate the total population based on the overlap generated by multiple captures [13]–[16]. ARIMA model is commonly used to model time series data to forecast the future trend [17]–[20]. ISENSE predicts two test completion indicators in an incremental manner, including: 1) total number of bugs predicted with CRC model, and 2) required test cost for achieving certain test objectives predicted with ARIMA model.

ISENSE is evaluated using 218 tasks from one of the largest Chinese crowdtesting platforms. Results show that, the MRE of prediction (on total bugs, and required cost) are both below 6%, with about 10% standard deviation.

We further demonstrate its effectiveness through two typical decision scenarios, one for automating task closing decision, and the other for semi-automation of task closing trade-off analysis. The results show that decision automation using ISENSE will provide managers with greater opportunities to achieve cost-effectiveness gains of crowdtesting. Specifically, a median of 100% bugs can be detected with 30% saved cost based on the automated close prediction.

The contributions of this paper are as follows:

- Empirical observations on crowdtesting bug arrival patterns based on industrial dataset, which has motivated this study and can motivate future studies.
- Integration of incremental sampling technique to model crowdtesting bug arrival data.
- Development of CRC-based model for predicting total number of bugs, and ARIMA-based model for predicting required cost for achieving certain test objectives.
- ISENSE approach for automated decision support in crowdtesting management, including automating task closing decision, and semi-automation of task closing trade-off analysis.
- Evaluation of ISENSE on 46,434 reports of 218 crowdtesting tasks from one of the largest crowdtesting platforms in China, and results are promising.

II. Background and Motivation

A. Background

In general crowdtesting practice, managers prepare the crowtesting task (including the software under test and test requirements), and distribute it on certain online crowdtesting platform. Crowdworkers can sign in their interested tasks and submit crowdtesting reports, typically summarizing test input, test steps, test results, etc. Managers usually set up either a fixed period (e.g., 5 days) or a fixed number of participant (e.g., recruiting 400 crowd workers) for the close criteria of crowdtesting task. If either of the criteria is met first, then the task will be automatically closed. There are different payout schema in crowdtesting, e.g., pay by report (see Section VI-C for details). Generally, the cost of a task is positively correlated with the number of received reports, thus with the close time.

The crowdtesting platform receives and manages crowdtesting reports submitted by the crowdworkers. Project managers then inspect and verify each report for their tasks manually or using automatic tool support (e.g., [21], [22] for report labeling). Generally, each report will be characterized using two attributes: 1) whether it contains a valid bug[2]; 2) if yes, whether it is a duplicate bug that has been previously reported in other reports. In the following paper, if not specified, when we say "*bug*" or "*unique bug*", we mean the corresponding report contains a bug and the bug is not the duplicate of previously submitted ones.

B. Baidu CrowdTest DataSet

Our experimental dataset is collected from Baidu[3] crowdtesting platform, which is one of the largest platforms in China. The dataset contains all tasks completed between May. 1st 2017 and Jul. 1st 2017. In total, there are 218 mobile application testing tasks from various domains[4], with 46434 submitted reports. The minimum, average, and maximum number of crowdtesting reports (*and unique bugs*) per task are 101 (*8*), 175 (*23*), and 798 (*99*), respectively.

[2]In our experimental platform, a report corresponds to either 0 or 1 bug, and there is no report containing more than 1 bug.

[3]test.baidu.com

[4]Details of dataset are in https://github.com/wangjunjieISCAS/CM

C. Observations From A Pilot Study

To understand the bug arrival patterns of crowdtesting, we conduct a pilot study to analyze three metrics, i.e. *bug detection speed*, *bug detection cost*, and *bug detection rate*.

For each task, we first identify the time when K% bugs was detected, where K is ranged from 10 to 100. Then, the *bug detection speed* for a task can be derived using the duration (*hours*) between its open time and the time it receives K% bugs. The *bug detection cost* can be derived using the number of submitted reports by reaching K% bugs.

To examine *bug detection rate*, we break the crowdtesting reports for each task into 10 equal-sized groups, in chronological order. The rate for each group is derived using the ratio between the number of unique bugs and the number of reports. In addition, for each crowdtesting task, we also examine the percentage of accumulated bugs (denoted as bug arrival curve) for the previous X reports, where X ranges from 1 to the total number of reports.

The following bug arrival patterns are observed:

1) ***Large Variation in Bug Detection Speed and Cost:*** Figure 1a and 1b demonstrates the distribution of bug detection speed and cost for all tasks. In general, there is large variation in bug detection speed and cost. Specifically, to achieve the same K% bugs, there is large variation in both metrics. This is particularly true for a larger K%. For example, when detecting 90% bugs, the bug detection cost ranges from 3 to 149 hours, and from 27 to 435 reports.

2) ***Decreasing Bug Detection Rates Over Time:*** Figure 1c shows the bug detection rate of the 10 break-down groups across all tasks. We can see that the bug detection rate decreases sharply during the crowdtesting process. This signifies that the cost-effectiveness of crowdtesting is dramatically decreasing for the later part of the process.

3) ***Plateau Effect of Bug Arrival Curve:*** Figure 2 shows typical bug arrival curves for four crowdtesting tasks. While they differ, somewhat, note that they all exhibit the same "plateau effect", after which (i.e., red point in Figure 2) new reports find no new bugs.

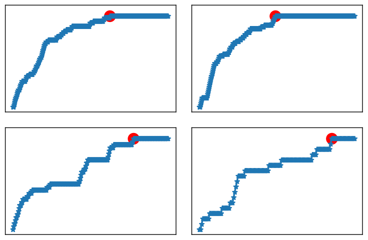


Figure 2: *Observations on real-world crowdtesting data - Bug arrival curve*

We assume the cost spent on these reports after the red point are wasteful spending. In the 218 experimental tasks, there is an average of 32% wasteful spending. The plateau effect together with the large amount of wasteful spending further suggest the potential opportunity and practical need for introducing early closing mechanism (based on the recognition of that plateau) to increase cost-effectiveness.

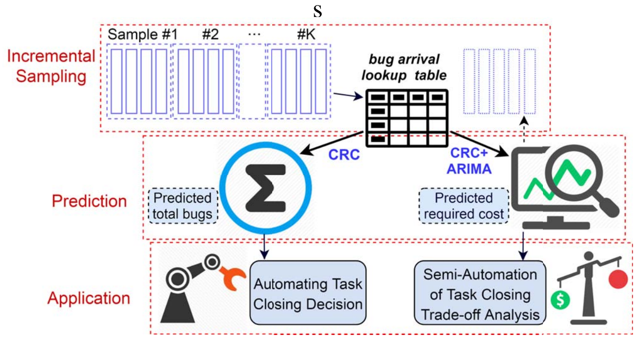


Figure 3: Overview of iSENSE

4) ***Needs of Automated Decision Support:*** In addition, an unstructured interview was conducted with the managers of Baidu, with findings shown below.

Project managers commented the black-box nature of crowdtesting process. While receiving constantly arriving reports, they are often clueless about the latent bugs, or the required cost to find them. Due to lack of situation awareness, the management of crowdtesting is conducted as a guesswork. This frequently results in many blind decisions in task planning and management. Besides, managers typically need to handle large number of crowdtesting tasks simultaneously, which is very labor intensive and error-prone in manual planning and management.

In Summary, because there is large variation in bug arrival speed and cost (Section II-C1), current decision making is largely done by guesswork. This results in low cost-effectiveness of crowdtesting (Section II-C2 and II-C3). A more effective alternative to manage crowdtesting would be to dynamically monitor the crowdtesting process and provide actionable decision support for task closing to save unnecessary cost wasting on later arriving reports. Besides, current practice suggests a practical need to empower managers with greater visibility into the crowdtesting processes (Section II-C4), and ideally raise their awareness about task progress, thus facilitate their decision making.

III. Approach

Figure 3 presents an overview of iSENSE. It consists of three main steps. First, iSENSE adopts an incremental sampling process to model crowdtesting reports. During the process, iSENSE converts the raw crowdtesting reports arrived chronologically into groups and generates a *bug arrival lookup table* to characterize the bug arrival information. Then, iSENSE integrates two models, i.e. CRC and ARIMA, to predict the total number of bugs contained in the software, and the required cost for achieving certain test objectives, respectively. Finally, iSENSE applies such estimates to support two typical crowdtesting decision scenarios, i.e., automating task closing decision, and semi-automation of task closing trade-off analysis. We will present each of the above steps in more details.

A. Preprocess Data With Incremental Sampling Technique

Incremental sampling technique [23] is a composite sampling and processing protocol. Its objective is to obtain a single sample for analysis that has an analytic concentration representative of the decision unit. It improves the reliability and defensibility of sampling data by reducing variability when compared to conventional discrete sampling strategies.

Considering the submitted crowdtesting reports of chronological order (Section II-A), when *smpSize* (*smpSize* is an input parameter) reports are received, iSENSE treats it as a representative group to reflect the multiple parallel crowdtesting sessions. Remember in Section II-A, we mentioned that, each report is characterized as: 1) whether it contains a bug; 2) whether it is duplicate of previously submitted reports; if no, it is marked with a new tag; if yes, it is marked with the same tag as the duplicates. During the crowdtesting process, we dynamically maintain a two-dimensional *bug arrival lookup table* to record these information.

Table I: Example of bug arrival lookup table

	#1	**#2**	**#3**	**#4**	**#5**	**#6**	**#7**	**#8**	**#9**	**#10**	**#11**	**#12**	...
Sample #1	1	1	1	0	0	0	0	0	0	0	0	0	
Sample #2	0	0	1	1	0	0	0	0	0	0	0	0	
Sample #3	0	0	1	0	1	0	0	0	0	0	0	0	
Sample #4	0	0	0	1	1	1	1	1	0	0	0	0	
Sample #5	0	0	1	1	0	0	0	1	1	1	1	0	
Sample #6	1	0	1	0	1	0	0	0	0	0	0	1	
Sample #7	...												

Table I provides an illustrative example. After each sample is received, we first add a new row (suppose it is row *i*) in the lookup table. We then go through each report contained in this sample. For the reports not containing a bug, we ignore it. Otherwise, if it is marked with the same tag as existing unique bugs (suppose it is column *j*), record *1* in row *i* column *j*. If it is marked with a new tag, add a new column in the lookup table (suppose it is column *k*), and record *1* in row *i* column *k*. For the empty cells in row *i*, fill it with 0.

B. Predict Total Bugs Using CRC

*1) **Background about CRC**:* The CRC (Capture-ReCapture) model was firstly used to estimate the size of an animal population in biology [24]–[27]. In doing so, animals are captured, marked and released on several trapping occasions. The number of marked animals that are recaptured allows one to estimate the total population size based on the samples overlap. It has also been applied in software inspections to estimate the total number of bugs [13]–[16]. Existing CRC models can be categorized into four types according to bug detection probability (i.e. identical vs. different) and crowdworker's detection capability (i.e. identical vs. different), as shown in Table II.

Model *M0* supposes all different bugs and crowdworkers have the same detection probability. Model *Mh* supposes that the bugs have different probabilities of being detected. Model *Mt* supposes that the crowdworkers have different detection capabilities. Model *Mth* supposes different detection probabilities for different bugs and crowdworkers.

Table II: Capture-ReCapture models

		Crowdworker's detection capability	
		Identical	Different
Bug detection probability	Identical	M0 (*M0*)	Mt (*MtCH*)
	Different	Mh (*MhJK, MhCH*)	Mth (*Mth*)

Based on the four basic CRC models, various estimators were developed. According to a recent systematic review [14], *MhJK* [25], *MhCH* [26], *MtCH* [27] are the three most frequently investigated and most effective estimators in software engineering. Apart from that, we investigate another two estimators (i.e., *M0* [24] and *Mth* [28]) to ensure all four basic models are investigated.

*2) **How to Use in iSENSE**:* iSENSE treats each sample as a capture (or recapture). At the end of each capture, after updating the *bug arrival lookup table*, iSENSE predicts the total number of bugs in the software[5] based on current lookup table. We only demonstrate how it works with *Mth* estimator due to space limit. For other four estimators, one can obtain the estimated bugs in a similar way[6].

Mth estimator predicts the total number of bugs based on Equation 1, 2 [28]. Table III shows the meaning of each variable, how to compute its value based on the bug arrival lookup table in Table I.

$$N = \frac{D}{C} + \frac{f_1}{C}\gamma^2,\ C = 1 - \frac{f_1}{\sum_{k=1}^{t} k f_k} \tag{1}$$

$$\gamma^2 = max\{\frac{\frac{D}{C}\sum_k k(k-1)f_k}{2\sum\sum_{j<k} n_j n_k} - 1, 0\} \tag{2}$$

Table III: Variables meaning and computation

Var.	**Meaning**	**Computation based on bug arrival lookup table**	**Example value**
N	Predicted total number of bugs		predicted value: ***24***
D	Actual number of bugs captured so far	Number of columns	12
t	Number of captures	Number of rows	6
n_j	Number of bugs detected in each capture	Number of cells with *1* in row *j*	3, 2, 2, 5, 6, 4
f_k	Number of bugs captured exactly *k* times in all captures, i.e., $\sum f_i = D$	Count the number of cells with *1* in each column, and denote as r_i; f_k is the number of r_i with value *k*	1=7, 2=2, 3=2, 5=1

C. Predict Required Cost Using ARIMA

*1) **Background about ARIMA**:* ARIMA (Autoregressive Integrated Moving Average) model is commonly used to model time series data to forecast the future values [17]–[20]. It extends ARMA (Autoregressive Moving Average) model by allowing for non-stationary time series to be modeled, i.e., a time series whose statistical properties such as mean, variance, etc. are not constant over time.

[5]To be precise, what we predict is the total number of potential bugs that are uncovered by crowdtesting.

[6]Refer to https://github.com/wangjunjieISCAS/CM for more details.

A time series is said to be autoregressive moving average (ARMA) in nature with parameters (p, q), if it takes the following form:

$$y_t = \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} + \epsilon_t \quad (3)$$

Where y_t is the current stationary observation, y_{t-i} for $i = 1, ..., p$ are the past stationary observations, ϵ_t is the current error, and ϵ_{t-i} for $i = 1, ..., q$ are the past errors. If this original time series $\{z_t\}$ is non-stationary, then d differences can be done to transform it into a stationary one $\{y_t\}$. These differences can be viewed as a transformation denoted by $y_t = \nabla^d z_t$, where $\nabla^d = (1 - B)^d$ where B is known as a backshift operator. When this differencing operation is performed, it converts an ARMA model into an ARIMA model with parameters (p, q, d).

*2) **How to Use in* ISENSE**:* Figure 4 demonstrates how ARIMA is applied in predicting future trend of bug arrival. We treat the reports of each sample as a window, and obtain the number of unique bugs submitted in each sample from the bug arrival lookup table (i.e., number of cells in the row whose value is *1* and the corresponding column does not have other *1*). Then we use the former *trainSize* windows to fit the ARIMA model and predict the number of bugs for the later *predictSize* windows. When new window is formed with the newly-arrived reports, we move the window by 1 and obtain the newly predicted results.

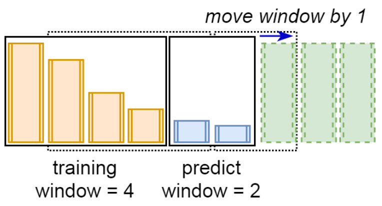


Figure 4: Illustrative example of ARIMA

Suppose one want to know how much extra cost is required for achieving certain test objective (i.e., *X*% bugs). As we already know the predicted total number of bugs (Section III-B), we can figure out how many bugs should be detected in order to meet the test objective; suppose it is *Y* bugs. Based on the prediction of ARIMA, we then obtain when the number of *Y* bugs can be received, suppose it needs extra K_i reports. In this way, we assume K_i is the required cost for meeting the test objective.

D. Apply ISENSE *to Two Decision Scenarios*

To demonstrate the usefulness of ISENSE, we generalize two typical decision scenarios in crowdtesting management, and illustrate its application to each scenario.

*1) **Automating Task Closing Decision**:* The first scenario that can benefit from the prediction of total bugs of ISENSE (Section III-B) is decision automation of dynamic task closing.

As soon as a crowdtesting task begins, ISENSE can be applied to monitor the actual bug arrival, constantly update the bug arrival lookup table, as well as keep tracking of the percentage of bugs detected (i.e., the ratio of the number of submitted bugs so far over the predicted total bugs).

In such scenario, different task close criteria can be customized in ISENSE so that it automatically closes the task when the specified criterion is met. For instance, a simple criterion would be to close the task when 100% bugs have been detected in submitted reports. Under this criterion, when ISENSE monitors 100% bugs have received and the prediction remains unchanged for successive two captures, it determines the time, when the last report was received, as the close time; and would automatically close the crowdtesting task at run time. Note that the restriction of two successive captures is to ensure the stability of the prediction.

ISENSE supports flexible customization of the close criteria. As an example, a task manager can set to close his/her tasks when 80% bugs have been detected. Consequently, ISENSE will help to monitor and close the task by reacting to these customized close criteria.

*2) **Semi-Automation of Task Closing Trade-off Analysis**:* The second scenario that benefits from the prediction of required cost of ISENSE (Section III-C) is decision support of task closing trade-off analysis.

For example, suppose 90% bugs have been reported at certain time, ISENSE can simultaneously reveal the estimated required cost for detecting an additional X% bugs (i.e., 5%), in order to achieve a higher bug detection level. Such cost-benefit related insights can provide managers with more confidence in making informed, actionable decision on whether to close immediately, if the required cost is too high to be worthwhile for additional X% detected bugs; or wait a little longer, if the required cost is acceptable and additional X% detected bugs is desired.

IV. Experiment Design

A. Research Questions

Four research questions are formulated to investigate the performance of the proposed ISENSE.

The first two research questions are centered around accuracy evaluation of the prediction of total bugs and required cost. Presumably, to support practical decision making, these underlying predictions should achieve high accuracy.

- RQ1: To what degree can ISENSE accurately predict total bugs?
- RQ2: To what degree can ISENSE accurately predict required cost to achieve certain test objectives?

The next two research questions are focused on investigating the effectiveness of applying ISENSE in the two typical scenarios (Section III-D), in which ISENSE is expected to facilitate current practices through automated and semi-automated decision support in managing crowdtesting tasks.

- RQ3: To what extent can ISENSE help increase the effectiveness of crowdtesting through decision automation?

- RQ4: How iSENSE can be applied to facilitate the trade-off decisions about cost-effectiveness?

B. Evaluation Metrics

We measure the **accuracy** of prediction based on Magnitude of Relative Error, a.k.a. ***MRE***, which is the most commonly-used measure for accuracy [29], [30]. It measures the relative error ratio between the actual value and predicted value, expressed as follows:

$$MRE = \frac{|actual\ value - predicted\ value|}{actual\ value} \tag{4}$$

It is applied in the prediction of total number of bugs (Section V-A) and required cost (Section V-B). Note that, although MRE has been argued as not the best metric for some applications as effort estimation [31], it is the most popular and widely-used metric [29], [30], so we use it in this work.

We measure the **cost-effectiveness** of close prediction (Section V-C) based on two metrics, i.e. bug detection level (i.e. %bug) and cost reduction (i.e. %reducedCost).

%bug is the percentage of bugs detected by the predicted close time. We treat the number of historical detected bugs as the total number. The larger *%bug*, the better.

%reducedCost is the percentage of saved cost by the predicted close time. To derive this metric, we first obtain the percentage of reports submitted at the close time, in which we treat the number of historical submitted reports as the total number. We suppose this is the percentage of consumed cost and %reducedCost is derived using 1 minus the percentage of consumed cost. The larger *%reducedCost*, the better.

Intuitively, an increase in %bug would be accompanied with a decrease in %reducedCost. Motivated by the F1 (or F-Measure) in prediction approaches of software engineering [21], [22], [32], we further derive an analogous metric ***F1***, to measure the harmonic mean of %bug and %reducedCost as follows:

$$F1 = \frac{2 \times \%bug \times \%reducedCost}{\%bug + \%reducedCost} \tag{5}$$

To further demonstrate the superiority of our proposed approach, we perform one-tailed Mann Whitney U test between our proposed iSENSE and baselines (Section IV-D). We include the Bonferroni correction to counteract the impact of multiple hypothesis tests. Besides the ***p-value*** for signifying the significance of the test, we also present the ***Cliff's delta*** to demonstrate the effect size of the test. We use the commonly-used criteria to interpret the effectiveness levels, i.e., *Large* (0.474-1.0), *Median* (0.33-0.474), *Small* (0.147-0.33), *Negligible* (-1, 0.147) (see details in [33]).

C. Experimental Setup

For RQ1, we set up 19 checkpoints in the range of receiving 10% to 100% reports, with an increment interval of 5% in between. At each checkpoint, we obtain the estimated total number of bugs at that time (see Section III-B). Based on the ground truth of actual total bugs, we then figure out MRE (Section IV-B) in predicting total bugs for each task.

For RQ2, we also set 19 checkpoints. Different from RQ1, the checkpoints of RQ2 is based on the percentage of detected bugs, i.e. from 10% bugs to 100% bugs with an increment of 5% in between. At each checkpoint, we predict the required test cost (Section III-C) to achieve an additional 5% bugs, i.e. target corresponding to the next checkpoint. For example, at checkpoint when 80% bugs have detected, we predict the required cost for achieving 85% bugs. Based on the ground truth of actual required cost, we then figure out MRE (Section IV-B) in predicting required cost.

For RQ3, we analyze the effectiveness of task closing automation with respect to five sample close criteria, i.e., close the task when 80%, 85%, 90%, 95%, or 100% bugs have detected, respectively. The reason why we choose these criteria is that we assume it is almost meaningless to close a task when less than 80% bugs being detected.

For RQ4, we use several illustrative cases from experimental projects to show how iSENSE can help trade-off decisions.

For all these experiments, we employ a commonly-used longitudinal data setup [21]. In detail, all the 218 tasks are sorted in the chronological order, and we use the former N-1 tasks as training set to tune the parameter (see details in Section IV-E) and use the N^{th} task as testing set to evaluate the performance of iSENSE. We experiment N from 19 to 218 to ensure a relative stable performance because a too small training set would bring in bias. In this way, we obtain the performance of 200 test tasks.

D. Baselines

we compare iSENSE with two baselines.

Rayleigh: This baseline is adopted from one of the most classical models for predicting the dynamic defect arrival in software measurement. Generally, it supposes the defect arrival data following the Rayleigh probability distribution [34]. In this experiment, we implement code to fit specific Rayleigh curve (i.e. the derived Rayleigh model) based on each task's bug arrival data, then predict the total bugs and required cost for certain test objectives (with the future bug trend) using the derived Rayleigh model.

Naive: This baseline is designed to employ naive empirical results, i.e. the median value of the dataset. Specifically, for the prediction of total bugs, it uses the median total bugs calculated based on the tasks of training set. For required cost, it uses the median required cost from training set, in terms of the corresponding checkpoint (Section IV-C).

E. Parameter Tuning

For each CRC estimator, the input parameter is *smpSize*, which represents how many reports are considered in each

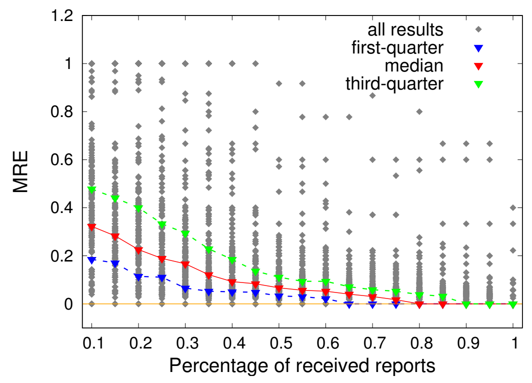


Figure 5: MRE of predicted total bugs of **Mth** (RQ1)

capture. We tune the optimal parameter value based on the training set (see Section IV-C) and apply it in the testing set to evaluate the performance. In detail, for every candidate parameter value (we experiment from 1 to 50) and for each checkpoint, we obtain the MRE for the prediction of total bugs for each task in the training set. We then calculate the median of these MRE values, and sum the median across all checkpoints for each candidate parameter value. We treat the parameter value, under which the smallest sum is obtained, as the best one. For ARIMA model, we use the same method for deciding the best parameter value.

V. Results and Analysis

A. Answers to RQ1 : Accuracy of Total Bugs Prediction

Table IV demonstrates the median and standard deviation for the *MRE* of predicted total bugs for all five CRC estimators. The columns correspond to different checkpoints, and the best two performer under each checkpoint are highlighted (in italic font and red color). We additionally present the detailed performance for *Mth* (the best estimator) in Figure 5[7].

From Table IV and Figure 5, we can see that, the predicted total number of bugs becomes more close to the actual total number of bugs (i.e., *MRE* decreases) towards the end of the tasks. Among the five estimators, *M0* and *Mth* have the smallest median *MRE* for most checkpoints. But the variance of *M0* is much larger than that of *Mth*. Hence, estimator *Mth* is more preferred because of its relatively higher stability and accurate prediction in total number of bugs. In the following experiments, if not specially mentioned, the results are referring to those generated from iSENSE with *Mth* estimator.

Comparison With Baselines: Table V compares the prediction accuracy of iSENSE and the two baselines, in terms of the median and standard deviation of *MRE*. Table VI summarizes the results of Mann-Whitney U Test for the *MRE* of predicted total bugs between each two methods. It shows that iSENSE significantly (p-value <0.05) and substantially (Cliff's delta is large) outperforms the two baselines, especially during the later stage (i.e. after the 40% checkpoint) of the crowdtesting tasks.

Answers to RQ1: iSENSE with the best estimator *Mth* is accurate in predicting the total bugs in crowdtesting, and significantly outperforms the two baselines. Specifically, the median of predicted total bugs is nearly equal with the ground truth (i.e., *MRE* <0.06) with a standard deviation of less than 10% during the latter half of the process.

[7]Detailed performance of other four estimators are in https://github.com/wangjunjieISCAS/CM

B. Answers to RQ2 : Accuracy of Required Cost Prediction

Table VII summarizes the comparison of median and standard deviation of the *MRE* of predicted required cost across iSENSE and the two baselines, with columns corresponding to different checkpoints. We highlight the methods with the best performance under each checkpoint. Table VIII presents the results of Mann-Whitney U Test between each pair.

As indicated by the decreasing median *MRE* in Table VII, the prediction of required cost becomes increasingly accurate for later checkpoints. For example, after 50% checkpoint, the median *MRE* of predicted cost is lower than 6%, with about 13% standard deviation. This implies that iSENSE can effectively predict the required cost to target test objectives.

Comparison With Baselines: We can see that the median and standard deviation of *MRE* for two baselines are worse than iSENSE during the second half of the task process. Observed from Table VIII, the difference between the proposed iSENSE and two baselines is significant (p-value <0.05) and substantial (Cliff's delta is not negligible) during the second half of crowdtesting process. This further signifies the advantages of the proposed iSENSE.

Answers to RQ2: iSENSE can predict the required test cost within averagely 6% *MRE* for later stage of crowdtesting.

C. Answers to RQ3: Task Closing Automation

Figure 6 shows the distribution of *%bug*, *%reducedCost*, and *F1* for five customized close criteria.

Let us first look at the last series of three boxes in Figure 6, which reflects a close criterion of 100% bugs being detected (i.e., most commonly-used setup). The results indicate that a median of 100% bugs can be detected with 30% median cost reduction. This suggests an additional 30% more cost-effectiveness for managers if equipped with such a decision automation tool as iSENSE to monitor and close tasks automatically at run-time. The reduced cost is a tremendous figure when considering the large number of tasks delivered in a crowdtesting platform. In addition, the standard deviation is relatively low, further signifying the stability of iSENSE.

We then shift our focus on other four customized close criteria (i.e., 80%, 85%, 90%, and 95% in terms of percentage of detected bugs). We can observe that for each

Table IV: Statistics for MRE of predicted total bugs (RQ1)

	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
	Median																		
M0	0.35	***0.25***	0.28	0.21	0.16	0.14	0.09	***0.07***	***0.07***	***0.05***	***0.04***	***0.03***	***0.03***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***
MtCH	0.41	0.32	0.28	***0.18***	***0.14***	0.13	0.09	***0.08***	0.08	0.06	0.05	0.04	0.03	0.03	0.03	***0.00***	***0.00***	***0.00***	***0.00***
MhCH	0.38	***0.25***	0.24	0.22	0.19	0.13	0.12	0.09	0.08	0.07	0.06	0.05	0.04	0.03	0.01	***0.00***	***0.00***	***0.00***	***0.00***
MhJK	***0.08***	***0.08***	***0.08***	***0.09***	***0.08***	***0.09***	***0.08***	0.09	0.11	0.10	0.09	0.09	0.10	0.10	0.08	0.07	0.06	0.05	0.04
Mth	***0.32***	0.28	***0.22***	0.18	0.16	***0.12***	***0.09***	0.08	***0.06***	***0.05***	***0.05***	***0.04***	***0.03***	***0.01***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***
	Standard deviation																		
M0	0.33	0.35	0.33	0.33	0.33	0.32	0.31	0.30	0.27	0.24	0.21	0.17	0.17	0.16	0.12	0.10	***0.06***	***0.07***	***0.04***
MtCH	0.33	0.32	0.31	0.30	0.30	0.27	0.24	0.19	0.16	***0.11***	0.13	***0.09***	***0.10***	***0.09***	***0.09***	***0.09***	0.09	0.09	0.05
MhCH	0.30	0.33	0.30	0.28	0.29	0.25	0.23	0.19	0.15	0.20	0.20	0.17	0.15	0.15	0.17	0.15	0.17	0.17	0.10
MhJK	***0.14***	***0.14***	***0.14***	***0.13***	***0.13***	***0.12***	***0.12***	***0.11***	***0.11***	0.12	***0.12***	0.12	0.12	0.13	0.13	0.12	0.12	0.10	0.07
Mth	***0.27***	***0.27***	***0.28***	***0.27***	***0.26***	***0.23***	***0.19***	***0.15***	***0.10***	***0.09***	***0.08***	***0.07***	***0.06***	***0.06***	***0.06***	***0.05***	***0.05***	***0.03***	***0.03***

Table V: Comparison with baselines in MRE of predicted total bugs (RQ1)

	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
	Median																		
ISENSE	0.32	***0.28***	***0.22***	***0.18***	***0.16***	***0.12***	***0.09***	***0.08***	***0.06***	***0.05***	***0.05***	***0.04***	***0.03***	***0.01***	***0.00***	***0.00***	***0.00***	***0.00***	***0.00***
Rayleigh	0.50	0.43	0.35	0.28	0.27	0.27	0.27	0.27	0.26	0.27	0.27	0.27	0.28	0.28	0.28	0.28	0.27	0.17	0.16
Naive	***0.29***	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.29	0.16
	Standard deviation																		
ISENSE	0.27	0.27	0.28	0.27	0.26	0.23	***0.19***	***0.15***	***0.10***	***0.09***	***0.08***	***0.07***	***0.06***	***0.06***	***0.06***	***0.05***	***0.05***	***0.03***	***0.03***
Rayleigh	***0.23***	***0.26***	***0.25***	***0.23***	***0.22***	***0.21***	0.24	0.26	0.28	0.40	0.40	0.51	0.54	0.58	0.58	0.55	0.41	0.39	0.29
Naive	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.40	0.20

Table VI: Results of Mann-Whitney U Test for MRE of predicted total bugs (RQ1)

	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
ISENSE vs. Rayleigh	0.00 *0.46*	0.00 *0.31*	0.00 *0.26*	0.00 *0.15*	0.00 *0.25*	0.00 *0.38*	0.00 *0.49*	0.00 *0.55*	0.00 *0.64*	0.00 *0.69*	0.00 *0.70*	0.00 *0.77*	0.00 *0.81*	0.00 *0.84*	0.00 *0.86*	0.00 *0.86*	0.00 *0.86*	0.00 *0.85*	0.00 *0.85*
ISENSE vs. Naive	0.86 *-0.0*	0.47 *0.00*	0.02 *0.11*	0.00 *0.17*	0.00 *0.28*	0.00 *0.40*	0.00 *0.50*	0.00 *0.57*	0.00 *0.66*	0.00 *0.72*	0.00 *0.73*	0.00 *0.78*	0.00 *0.82*	0.00 *0.84*	0.00 *0.87*	0.00 *0.89*	0.00 *0.90*	0.00 *0.89*	0.00 *0.79*
Rayleigh vs. Naive	1.0 *-0.5*	0.99 *-0.2*	0.99 *-0.1*	0.29 *0.03*	0.19 *0.05*	0.18 *0.05*	0.18 *0.05*	0.07 *0.08*	0.05 *0.09*	0.10 *0.07*	0.13 *0.06*	0.33 *0.02*	0.36 *0.01*	0.53 *-0.0*	0.53 *-0.0*	0.42 *0.01*	0.16 *0.05*	0.00 *0.15*	0.51 *-0.0*

Note that: The upper figure within a cell is p-value, and the lower figure is Cliff's delta. Background denotes the effect size of Large, Median, Small, and Negligible.

Table VII: Statistics and comparison with baselines for MRE of predicted required cost (RQ2)

	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
	Median																		
ISENSE	***0.33***	***0.25***	***0.15***	***0.13***	***0.13***	***0.10***	***0.08***	***0.06***	***0.07***	***0.06***	***0.04***	***0.03***	***0.04***	***0.04***	***0.05***	***0.04***	***0.05***	***0.06***	***0.06***
Rayleigh	1.01	0.66	0.37	0.33	0.25	0.18	0.14	0.11	0.12	0.10	0.08	0.07	0.08	0.08	0.07	0.08	0.10	0.17	0.14
Empirical	***0.33***	0.40	***0.15***	***0.13***	0.17	0.18	0.10	0.11	0.12	0.13	0.09	0.10	0.10	0.10	0.12	0.12	0.15	0.16	0.18
	Standard deviation																		
ISENSE	***0.79***	***0.57***	***0.36***	***0.48***	***0.26***	***0.24***	***0.20***	***0.22***	***0.12***	***0.13***	***0.13***	***0.12***	***0.11***	***0.13***	***0.13***	***0.08***	***0.11***	***0.13***	***0.11***
Rayleigh	1.33	0.93	0.63	0.70	0.39	0.33	0.28	0.29	0.17	0.34	0.30	0.34	0.27	0.38	0.40	0.35	0.38	0.40	0.20
Naive	***0.79***	0.76	***0.36***	***0.48***	0.33	0.33	0.24	0.29	0.17	0.20	0.17	0.16	0.15	0.19	0.22	0.14	0.18	0.30	0.19

Table VIII: Results of Mann-Whitney U Test for MRE of predicted required cost (RQ2)

	10%	15%	20%	25%	30%	35%	40%	45%	50%	55%	60%	65%	70%	75%	80%	85%	90%	95%	100%
ISENSE vs. Rayleigh	0.00 *0.38*	0.00 *0.38*	0.00 *0.34*	0.00 *0.37*	0.00 *0.33*	0.00 *0.30*	0.00 *0.33*	0.00 *0.29*	0.00 *0.28*	0.00 *0.27*	0.00 *0.34*	0.00 *0.33*	0.00 *0.36*	0.00 *0.30*	0.00 *0.27*	0.00 *0.30*	0.00 *0.32*	0.00 *0.31*	0.00 *0.25*
ISENSE vs. Naive	0.50 *0.0*	0.00 *0.19*	0.50 *0.0*	0.50 *0.0*	0.00 *0.16*	0.00 *0.30*	0.00 *0.18*	0.00 *0.29*	0.00 *0.28*	0.00 *0.45*	0.00 *0.41*	0.00 *0.48*	0.00 *0.45*	0.00 *0.38*	0.00 *0.46*	0.00 *0.43*	0.00 *0.40*	0.00 *0.36*	0.00 *0.49*
Rayleigh vs. Naive	0.99 *-0.3*	0.99 *-0.2*	0.99 *-0.3*	0.99 *-0.3*	0.99 *-0.1*	0.54 *-0.0*	0.99 *-0.1*	0.50 *0.0*	0.50 *0.0*	0.00 *0.17*	0.09 *0.07*	0.00 *0.13*	0.05 *0.08*	0.07 *0.07*	0.00 *0.14*	0.04 *0.09*	0.25 *0.03*	0.40 *0.01*	0.00 *0.19*

Note that: The upper figure within a cell is p-value, and the lower figure is Cliff's delta. Background denotes the effect size of Large, Median, Small, and Negligible.

close criterion, the median *%bug* generated from ISENSE is very close to the targeted close criterion, with small standard deviation. Among these close criteria, 36% to 52% cost can be saved, which further signify the effectiveness of ISENSE.

We also notice that, the median *%bug* is a little larger than the customized close criterion. For example, if the project manager hopes to close the task when 90% bugs detected, a median of 92% bugs have submitted at the predicted close time. This implies, in most cases, the close prediction produced by ISENSE does not have the risk of insufficient testing. Furthermore, we have talked with the project managers and they thought, detecting slightly more bugs (even with less reduced cost) is always better than detecting fewer bugs (with more reduced cost). This is because *%bug* is more like the constraint, while *%reducedCost* is only the bonus.

We also analyze the reason for this phenomenon. It is mainly because, before suggesting close, our approach requires the predicted total bugs remain unchanged for two successive captures (Section III-D1). This restriction is to alleviate the risk of insufficient testing. This is also because

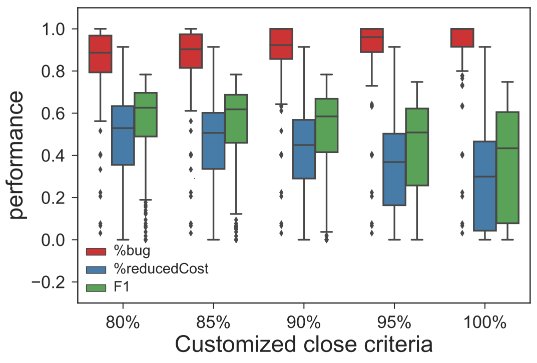


Figure 6: Task closing automation performance (RQ3)

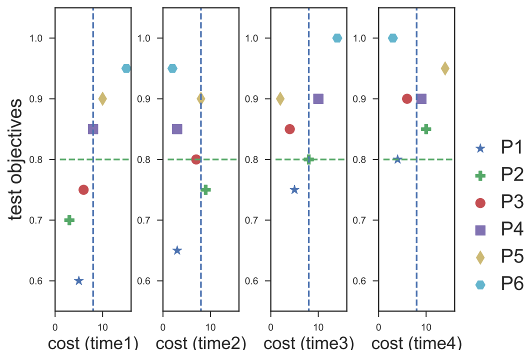


Figure 7: Decision support for trade-off analysis (RQ4)

we treat a sample of reports as the unit during the prediction, which can also potentially result in the close time being a little later than the customized close time.

Answers to RQ3: The automation of task closing by iSENSE can make crowdtesting more cost-effective, i.e., a median of 100% bugs can be detected with 30% saved cost.

D. Answers to RQ4: Trade-off Decision Support

Trade-off between investment and outcomes is important for optimizing the resource allocation. To reflect such trade-off context, we randomly pick a time and slice the experimental dataset to retrieve all tasks under testing at that time, then examine the cost-effectiveness of more testing.

Figure 7 demonstrates 4 trade-off analysis examples across 6 tasks (i.e. *P1-P6*), generated from repeating the above analysis at four different time points (i.e. corresponding to *time1 to time4* in a sequential order). The y-axis denotes the next test objective to achieve, while x-axis shows the predicted required cost to achieve that objective.

Generally speaking, in each of the four boxes, the crowdtesting tasks to the right are less cost-effective than the tasks to the left. For example, at *time3*, *P6* is estimated to require additional 14 cost to achieve 100% test objective. If the manager is facing budget constraints or trying to improve cost-effectiveness, he/she could choose to close *P6* at *time3*, because it is the least effective one among all tasks.

To facilitate such kind of trade-off analysis on which task to close and when to close, we design two decision parameters as inputs from decision maker: 1) *quality benchmark* which sets the minimal threshold for bug detection level, e.g. the horizontal red lines in Figure 7; 2) *cost benchmark* which sets the maximal threshold for required cost to achieve the next objective, e.g. the vertical blue lines in Figure 7.

These two benchmarks split the tasks into four regions at each slicing time (as indicated by the four boxes in each subfigure of Figure 7). Each region suggests different insights on the test sufficiency as well as cost-effectiveness for more testing, which can be used as heuristics to guide actionable decision-making at run time. More specifically:

Lower-Left (Close-Later): Tasks in this quadrant are recommended to be closed later, and continuing testing produces low hanging fruits. For these tasks, their quality levels are not acceptable yet, and it only requires relatively less cost to improve quality (i.e., to achieve next test objective). This indicates the most cost-effective options and testing should definitely continue.

Lower-Right (Continue-Manage): Tasks in this quadrant have not met the quality benchmark, so continue testing is preferred even though they require significant more cost to achieve quality objective. It also suggests that the task is either difficult to test, or the current crowdworker participation is not sufficient. Therefore, managers are recommended to take actions to drill down in particular tasks, and see if more testing guidelines or worker incentives are needed.

Upper-Left (Plan-to-close): Tasks in this quadrant already meet their quality benchmark, however, it is very cost-effectiveness for reaching next higher quality level, i.e, with little additional cost. Managers may plan to close all these tasks, or identify some high priority ones for further testing and quality improvement.

Upper-Right (Close-Now): Tasks in this quadrant are candidates for closing immediately, since they have meet the pre-specified quality objectives and will require relatively greater cost to reach next quality level. It is not practical to continue testing considering the cost-effectiveness.

Note that, the two benchmarks in Figure 7 can be customized according to practical needs.

Answers to RQ4: iSENSE provides practical insights to help managers make trade-off analysis on which task to close or when to close, based on two benchmark parameters and a set of decision heuristics.

VI. Discussion

A. Best CRC Estimator for Crowdtesting

In traditional software inspection or testing activities, *MhJK*, *MhCH*, and *MtCH* have been recognized as the most effective estimators for total bugs [13], [15], [16], [35]–[38]. However, in crowdtesting, the most comprehensive estimator *Mth* (see Section III-B1) outperforms the other CRC estimators. This is reasonable because crowdtesting is conducted by a diversified pool of crowdworkers with different levels of capability, experience, testing devices, and the nature of bugs in the software under test also vary greatly in terms of types, causes, and detection difficulty, etc. In such cases, *Mth*, which assumes different detection probability for both bugs and crowdworkers, supposes to be the most suitable estimator for crowdtesting.

B. Necessity for More Time-Sensitive Analytics in Crowdtesting Decision Support

As discussed earlier in the background and motivational pilot study (Section II-C), challenges associated with crowdtesting management mainly lie in two aspects: uncertainty in crowdworker's performance and lack of visibility into crowdtesting progress. We believe there is an increasing need for introducing more time-sensitive analytics during crowdtesting process to support better decision making.

ISENSE can generate time-based information revealing dynamic crowdtesting progress and provide practical guidelines of trade-off analysis. Besides, ISENSE provides additional visibility into the testing progress and insights for effective task management along with the crowdtesting process. The experimental results have proven that a significant portion of crowdtesting cost can be saved. This is extremely encouraging and we look forward to more discussion and innovative decision support techniques in this direction.

C. Threats to Validity

First, this paper treats the number of crowdtesting reports as the amount of cost when measuring the reduced cost (Section IV-B). This is applicable for the *paid by report* schema (i.e., *crowdworkers who submit report can get paid*) [2], which is a commonly-used payout schema. For a second popular schema *paid by bug* (i.e., *crowdworkers who report bug can get paid*) [3], [39], ISENSE can also reduce the cost by closing the task properly (i.e., fewer bug reports means less cost). And we believe ISENSE can obtain a comparable performance because the proportion of bugs in the reports almost remain unchanged across the crowdtesting process. For a third popular schema *paid by first bug* (i.e., *crowdworkers who report the first bug can get paid*) [4]. This schema is less sensitive to the automated task closing decision because under this schema, the payment to crowdworkers is constant (i.e., the submitted number of unique bugs). However, by closing the task at proper time, the platform can potentially reduce the cost for managing the duplicate reports, as well as shorten the duration of crowdtesting tasks.

Second, our designed methods are based on the report's attributes (i.e., whether it contains a bug; and whether it is the duplicates of previous ones). In crowdtesting process, each report would be inspected and triaged with these two attributes so as to better manage the reported bugs and facilitate bug fixing [2], [39]. This can be done manually or using automatic tool support (e.g., [21], [22]). Therefore, we assume our designed methods can be easily adopted in the crowdtesting platform.

VII. Related Work

Crowdtesting has been applied to facilitate many testing tasks, e.g., test case generation [40], usability testing [41], software performance analysis [42], software bug detection and reproduction [43]. These studies leverage crowdtesting to solve the problems in traditional testing activities, while some other approaches focus on solving the new encountered problems in crowdtesting. Feng et al. [44], [45] and Jiang et al. [46] proposed approaches to prioritize test reports in crowdtesting. Wang et al. [21], [22], [47] proposed approaches to automatically classify crowdtesting reports. Cui et al. [48], [49] and Xie et al. [50] proposed crowdworker selection approaches for crowdtesting tasks. This work focuses on the automated decision support for crowdtesting management, which is valuable to improve the cost-effectiveness of crowdtesting and not explored before.

Many existing approaches proposed risk-driven or value-based analysis to prioritize or select test cases [51]–[59], so as to improve the cost-effectiveness of testing. However, none of them is applicable to the emerging crowdtesting paradigm where managers typically have no control over online crowdworkers' dynamic behavior and uncertain performance. There are also existing researches focusing on defect prediction and effort estimation [30], [32], [60]–[62]. The core part of these approaches is the extraction of features from the source code, or software repositories. However, in crowdtesting, the platform can neither obtain the source code of these apps, nor involve in the development process of these apps. Several researches focused on studying the time series models for measuring software reliability [12], [63]–[68]. Among these, ARIMA is the most promising one for modeling software failures over time [17], [18], [20]. This paper used ARIMA in modeling the bug arrival dynamics in crowdtesting and estimating future trend.

VIII. Conclusion

Motivated by the empirical observations from an industry crowdtesting platform, we propose completion-aware crowdtesting management approach ISENSE which can raise the awareness of testing progress through two completion indicators, and be used to automate the task closing and semi-automate trade-off decisions.

Acknowledgments

This work is supported by the National Key Research and Development Program of China under grant No.2018YFB1401000, National Natural Science Foundation of China under grant No.61602450, No.6143200, No.61802374 and China Scholarship Council.

REFERENCES

[1] K. Mao, L. Capra, M. Harman, and Y. Jia, "A survey of the use of crowdsourcing in software engineering," *Journal of Systems and Software*, vol. 126, pp. 57–84, 2017.

[2] X. Zhang, Y. Feng, D. Liu, Z. Chen, and B. Xu, "Research progress of crowdsourced software testing," *Journal of Software*, vol. 29(1), pp. 69–88, 2018.

[3] http://www.softwaretestinghelp.com/crowdsourced-testing-companies/, 2018.

[4] https://dzone.com/articles/top-10-benefits-of-crowd-testing-1, 2018.

[5] https://www.softwaretestinghelp.com/guide-to-crowdsourced-testing/, 2018.

[6] https://dzone.com/articles/why-google-uses-crowd-testing, 2018.

[7] https://www.applause.com/customers/, 2018.

[8] https://www.gartner.com/doc/3472119/, 2018.

[9] G. J. Myers, C. Sandler, and T. Badgett, *The art of software testing*. John Wiley & Sons, 2011.

[10] W. E. Lewis, *Software testing and continuous quality improvement*. CRC press, 2016.

[11] M. Garg, R. Lai, and S. J. Huang, "When to stop testing: a study from the perspective of software reliability models," *IET software*, vol. 5, no. 3, pp. 263–273, 2011.

[12] J. Iqbal, N. Ahmad, and S. Quadri, "A software reliability growth model with two types of learning," in *2013 International Conference on Machine Intelligence and Research Advancement*. IEEE, 2013, pp. 498–503.

[13] G. Rong, B. Liu, H. Zhang, Q. Zhang, and D. Shao, "Towards confidence with capture-recapture estimation: An exploratory study of dependence within inspections," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*, 2017, pp. 242–251.

[14] G. Liu, G. Rong, H. Zhang, and Q. Shan, "The adoption of capture-recapture in software engineering: a systematic literature review," in *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering*. ACM, 2015, p. 15.

[15] Y. H.Chun, "Estimating the number of undetected software errors via the correlated capture-recapture model," *European Journal of Operational Research*, vol. 175, no. 2, pp. 1180 – 1192, 2006.

[16] N. R. Mandala, G. S. Walia, J. C. Carver, and N. Nagappan, "Application of kusumoto cost-metric to evaluate the cost effectiveness of software inspections," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 221–230.

[17] A. Amin, L. Grunske, and A. Colman, "An approach to software reliability prediction based on time series modeling," *Journal of Systems and Software*, vol. 86, no. 7, pp. 1923–1932, 2013.

[18] C. Chong Hok Yuen, "On analyzing maintenance process data at the global and the detailed levels: A case study," in *Proceedings of the IEEE Conference on Software Maintenance*, 1988, pp. 248–255.

[19] C. F. Kemerer and S. Slaughter, "An empirical approach to studying software evolution," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 493–509, 1999.

[20] I. Herraiz, J. M. Gonzalez-Barahona, and G. Robles, "Forecasting the number of changes in eclipse using time series analysis," in *Fourth International Workshop on Mining Software Repositories*. IEEE, 2007, pp. 32–32.

[21] J. Wang, Q. Cui, Q. Wang, and S. Wang, "Towards effectively test report classification to assist crowdsourced testing," in *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. ACM, 2016, p. 6.

[22] J. Wang, Q. Cui, S. Wang, and Q. Wang, "Domain adaptation for test report classification in crowdsourced testing," in *Proceedings of the 39th International Conference on Software Engineering: Software Engineering in Practice Track*. IEEE Press, 2017, pp. 83–92.

[23] L. Mora-Applegate and M. Malinowski, "Incremental sampling methodology," Interstate Technology and Regulatory Council (ITRC), Tech. Rep., 2012.

[24] P. S. Laplace, "Sur les naissances, les mariages et les morts," *Histaire de I'Academie Royale des Sciences*, p. 693, 1783.

[25] K. P. Burnham and W. S. Overton, "Estimation of the size of a closed population when capture probabilities vary among animals," *Biometrika*, vol. 65, no. 3, pp. 625–633, 1978.

[26] A. Chao, "Estimating animal abundance with capture frequency data," *The Journal of Wildlife Management*, pp. 295–300, 1988.

[27] ——, "Estimating the population size for capture-recapture data with unequal catchability," *Biometrics*, pp. 783–791, 1987.

[28] S.-M. Lee, "Estimating population size for capture-recapture data when capture probabilities vary by time, behavior and individual animal," *Communications in Statistics-Simulation and Computation*, vol. 25, no. 2, pp. 431–457, 1996.

[29] S. D. Conte, H. E. Dunsmore, and Y. Shen, *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[30] E. Kocaguneli, T. Menzies, and J. W. Keung, "On the value of ensemble effort estimation," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1403–1416, 2012.

[31] D. Port and M. Korte, "Comparative studies of the model evaluation criterions mmre and pred in software cost estimation research," in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. ACM, 2008, pp. 51–60.

[32] J. Nam, W. Fu, S. Kim, T. Menzies, and L. Tan, "Heterogeneous defect prediction," *IEEE Transactions on Software Engineering*, 2017.

[33] N. Cliff, *Ordinal methods for behavioral data analysis*. Psychology Press, 2014.

[34] S. H. Kan, *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[35] L. C. Briand, K. E. Emam, B. G. Freimut, and O. Laitenberger, "A comprehensive evaluation of capture-recapture models for estimating software defect content," *IEEE Transactions on Software Engineering*, vol. 26, no. 6, pp. 518–540, 2000.

[36] G. S. Walia and J. C. Carver, "Evaluating the effect of the number of naturally occurring faults on the estimates produced by capture-recapture models," in *2009 International Conference on Software Testing Verification and Validation*, 2009, pp. 210–219.

[37] A. Goswami, G. Walia, and A. Singh, "Using learning styles of software professionals to improve their inspection team performance," *International Journal of Software Engineering and Knowledge Engineering*, vol. 25, no. 09-10, pp. 1721–1726, 2015.

[38] P. Vitharana, "Defect propagation at the project-level: results and a post-hoc analysis on inspection efficiency," *Empirical Software Engineering*, vol. 22, no. 1, pp. 57–79, 2017.

[39] 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 Software Engineering*, pp. 1–25, 2018.

[40] N. Chen and S. Kim, "Puzzle-based automatic testing: Bringing humans into the loop by solving puzzles," in *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. ACM, 2012, pp. 140–149.

[41] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. da Rocha, and T. M. G. d. A. Barbosa, "Affective crowdsourcing applied to usability testing," *Int. J. of Computer Science and Information Technologies*, vol. 5, no. 1, pp. 575–579, 2014.

[42] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, and S. Ganguly, "Leveraging the crowd: How 48,000 users helped improve lync performance," *IEEE Software*, vol. 30, no. 4, pp. 38–45, 2013.

[43] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier, "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring," in *2016 IEEE/ACM International Conference on Mobile Software Engineering and Systems*. IEEE, 2016, pp. 88–99.

[44] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced testing." in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 225–236.

[45] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective test report prioritization using image understanding," in *2016 31st IEEE/ACM International Conference on Automated Software Engineering*. IEEE, 2016, pp. 202–213.

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

[47] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based active classification of test report to assist crowdsourced testing," in *2016 31st International Conference on Automated Software Engineering*. IEEE, 2016, pp. 190–201.

[48] Q. Cui, J. Wang, G. Yang, M. Xie, Q. Wang, and M. Li, "Who should be selected to perform a task in crowdsourced testing?" in *2017 IEEE 41st Annual Computer Software and Applications Conference*, vol. 1. IEEE, 2017, pp. 75–84.

[49] Q. Cui, S. Wang, J. Wang, Y. Hu, Q. Wang, and M. Li, "Multi-objective crowd worker selection in crowdsourced testing," in *29th International Conference on Software Engineering and Knowledge Engineering*, 2017, pp. 218–223.

[50] M. Xie, Q. Wang, G. Yang, and M. Li, "Cocoon: Crowdsourced testing quality maximization under context coverage constraint," in *2017 IEEE 28th International Symposium on Software Reliability Engineering*. IEEE, 2017, pp. 316–327.

[51] S. Wang, J. Nam, and L. Tan, "QTEP: quality-aware test case prioritization," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 2017, pp. 523–534.

[52] A. Shi, T. Yung, A. Gyori, and D. Marinov, "Comparing and combining test-suite reduction and regression test selection," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 237–247.

[53] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke, "Empirical evaluation of pareto efficient multi-objective regression test case prioritisation," in *Proceedings of the 2015 International Symposium on Software Testing and Analysis*. ACM, 2015, pp. 234–245.

[54] R. K. Saha, L. Zhang, S. Khurshid, and D. E. Perry, "An information retrieval approach for regression test prioritization based on program changes," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 268–279.

[55] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon, "Comparing white-box and black-box test prioritization," in *2016 IEEE/ACM 38th International Conference on Software Engineering*. IEEE, 2016, pp. 523–534.

[56] A. Panichella, R. Oliveto, M. Di Penta, and A. De Lucia, "Improving multi-objective test case selection by injecting diversity in genetic algorithms," *IEEE Transactions on Software Engineering*, vol. 41, no. 4, pp. 358–383, 2015.

[57] K. Wang, C. Zhu, A. Celik, J. Kim, D. Batory, and M. Gligoric, "Towards refactoring-aware regression test selection," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18, 2018, pp. 233–244.

[58] L. Zhang, "Hybrid regression test selection," in *Proceedings of the 40th International Conference on Software Engineering*, 2018, pp. 199–209.

[59] B. Miranda, E. Cruciani, R. Verdecchia, and A. Bertolino, "Fast approaches to scalable similarity-based test case prioritization," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18, 2018.

[60] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE transactions on software engineering*, vol. 33, no. 1, pp. 2–13, 2007.

[61] A. Agrawal and T. Menzies, """better data" is better than "better data miners" (benefits of tuning smote for defect prediction)," in *Proceedings of the 40th International Conference on Software engineering*, 2018.

[62] P. K. Singh, D. Agarwal, and A. Gupta, "A systematic review on software defect prediction," in *2015 2nd International Conference on Computing for Sustainable Global Development*. IEEE, 2015, pp. 1793–1797.

[63] D. Zeitler, "Realistic assumptions for software reliability models," in *Software Reliability Engineering, 1991. Proceedings., 1991 International Symposium on*. IEEE, 1991, pp. 67–74.

[64] C. Bai, Q. Hu, M. Xie, and S. H. Ng, "Software failure prediction based on a markov bayesian network model," *Journal of Systems and Software*, vol. 74, no. 3, pp. 275–282, 2005.

[65] N. Fenton, M. Neil, and D. Marquez, "Using bayesian networks to predict software defects and reliability," *Proceedings of the Institution of Mechanical Engineers, Part O: Journal of Risk and Reliability*, vol. 222, no. 4, pp. 701–712, 2008.

[66] M. Wiper, A. Palacios, and J. Marín, "Bayesian software reliability prediction using software metrics information," *Quality Technology & Quantitative Management*, vol. 9, no. 1, pp. 35–44, 2012.

[67] B. Yang, X. Li, M. Xie, and F. Tan, "A generic data-driven software reliability model with model mining technique," *Reliability Engineering & System Safety*, vol. 95, no. 6, pp. 671–678, 2010.

[68] M. das Chagas Moura, E. Zio, I. D. Lins, and E. Droguett, "Failure and reliability prediction by support vector machines regression of time series data," *Reliability Engineering & System Safety*, vol. 96, no. 11, pp. 1527–1534, 2011.