

Article

# Collaborative Crowdsourced Software Testing

Sultan Alyahya 

Department of Information Systems, King Saud University, Riyadh 11451, Saudi Arabia; sualyahya@ksu.edu.sa

**Abstract:** Crowdsourced software testing (CST) uses a crowd of testers to conduct software testing. Currently, the microtasking model is used in CST; in it, a testing task is sent to individual testers who work separately from each other. Several studies mentioned that the quality of test reports produced by individuals was a drawback because a large number of invalid defects were submitted. Additionally, individual workers tended to catch the simple defects, not those with high complexity. This research explored the effect of having pairs of collaborating testers working together to produce one final test report. We conducted an experiment with 75 workers to measure the effect of this approach in terms of (1) the total number of unique valid defects detected, (2) the total number of invalid defects reported, and (3) the possibility of detecting more difficult defects. The findings show that testers who worked in collaborating pairs can be as effective in detecting defects as an individual worker; the differences between them are marginal. However, CST significantly affects the quality of test reports submitted in two dimensions: it helps reduce the number of invalid defects and also helps detect more difficult defects. The findings are promising and suggest that CST platforms can benefit from new mechanisms that allow for the formation of teams of two individuals who can participate in doing testing jobs.

**Keywords:** crowdsourcing; software testing; crowdsourced software testing; collaboration; collaborative software engineering; tester performance



**Citation:** Alyahya, S. Collaborative Crowdsourced Software Testing. *Electronics* **2022**, *11*, 3340. <https://doi.org/10.3390/electronics11203340>

Academic Editors: Junjun Zheng, Tadashi Dohi and Hiroyuki Okamura

Received: 10 September 2022

Accepted: 14 October 2022

Published: 17 October 2022

**Publisher's Note:** MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.



**Copyright:** © 2022 by the author. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

## 1. Introduction

Crowdsourced software testing (CST) is an emerging area in the domain of software engineering. It is a process in which software testing activities are carried out by a crowd, which is commonly a large and undefined group of heterogeneous individuals [1]. CST supports the production of high-quality software by engaging thousands of people in the testing of software under thousands of conditions. This approach can support traditional software testing, which normally involves only a few testers working inside a software development organization [2].

CST has been successful in several types of testing, such as usability testing, functional testing, and performance testing. It can be conducted using workflows such as those provided by uTest and Figure Eight. These platforms receive requests from clients (e.g., companies that need software testing services) and distribute these requests to the crowd registered with the platform, which is waiting to perform testing tasks.

The current practice in CST is to use the microtasking model, that is, a model in which software testing tasks are sent to individual testers who work separately from each other [3]. In microtasking, all testers have the same aim of finding defects in the software given to them. This can result in maximizing the total number of defects found.

However, several studies mentioned that the quality of the test reports produced by individuals is a drawback because a large number of invalid defects is submitted. Additionally, individual workers tend to catch the simple defects, not those with high complexity [4–6].

This research explores the effect of having pairs of collaborating workers performing testing on the Internet and producing one final test report. In particular, we studied whether

collaborating pairs outperformed isolated individuals in terms of (1) the total number of unique valid defects detected, (2) the total number of invalid defects reported by workers, and (3) the possibility of detecting more difficult defects rather than concentrating only on detecting simple defects. The result could be supportive or could negate the concept of using collaborating pairs for CST.

The study was carried out using an experiment involving 75 testers who were divided into groups of one or two workers and asked to do exploratory functional testing.

The remainder of this paper is divided as follows: Section 2 provides a literature review about the research topic, and Section 3 describes the approach used in this study in detail. Section 4 presents the findings of the study, while Section 5 discusses these findings. Finally, Section 6 concludes the paper and suggests possible future research directions.

## 2. Literature Review

CST is the process of using a crowd of testers to conduct software testing [7]. It has been used in many types of testing, such as performance testing [8], functional testing [9], usability testing [10], and compatibility testing [11]. The crowd can be a group recruited by intermediary platforms, a group of internal employees, or even a group of customers.

The current practice in CST is to let members of the crowd work individually on the same testing job [3]. As Linus Law states, “given enough eyeballs, all bugs are shallow” [12]; hence, the idea behind adding multiple testers is to increase the likelihood of detecting more defects.

Researchers found several reasons why software testing could be seen as an additive task [13] (i.e., why adding more workers would improve the final output). A main reason was the limited attention of individuals [14]. Fry and Weimer [15] found that human testers were able to observe certain types of defects more than others. For instance, human testers were five times as accurate at identifying extra statements as missing statements. Another factor was the differences in expertise and capability among testers; for instance, three beginner testers might find the same number of usability issues as one expert tester [16].

In [14], the researchers assessed the value of having several individuals test a manual system of the same function. They stated that an increase in the number of individuals doing the testing increased the crowd’s ability to detect a defect. However, it also caused a negative effect, because the relative share of invalid defect reports and duplicated test reports increased. The above research studies focused on situations in which multiple testers worked independently from each other and each person produced a test report.

CSTs tend to use the microtasking model to do testing tasks [17]. In this model, a particular task is divided into small microtasks that can be completed by individuals. It is most appropriate for simple jobs that require minimal effort, but it may not work perfectly for complex tasks such as software testing [18]. Software systems are sophisticated structures with highly interrelated parts, and they may not be easily broken down into clearly articulated and self-contained testing tasks [3].

Another aspect of software testing is that it requires multiple activities, including planning, execution, and evaluation [19]. Test planning involves planning the testing coverage, selecting the test cases, and determining the testing steps. Test execution is the running of the test cases selected during test planning. Finally, in test evaluation, the test results are analyzed, the identified defects are evaluated, and an assessment is made regarding whether further testing is required [20]. This process is complex, especially with exploratory functional testing.

In spite of the fact that the current form of CST does not consider collaboration between testers, the idea of having teams in software testing is not new. Many of the software development organizations have their own testing teams, who divide the work between themselves and cooperatively help each other to discover defects [21]. It is widely recommended that testing be done in pairs and that others be engaged to obtain critiques and advice [22]. Nevertheless, to the best of our knowledge, there is no research on the comparison of testing teams with isolated testers, neither in laboratory settings nor CST contexts.

If we consider the generic domain of crowdsourcing, we notice that it has an overall aim similar to that of CST. It strives to produce higher-quality results and reduce wrong or low-quality results, especially in complex tasks. There have been increasing research studies focusing on task assignments to teams. These studies aimed to find the best group of workers to complete a task. For example, task assignment algorithms were developed to combine different human factors, such as workers' skills and worker-to-worker affinity [23]. Skills are used to remove unqualified workers, while worker-to-worker affinity attempts to find the "comfort level" of workers who are part of a team solving a collaborative task [24]. Although these studies supported the argument that complex tasks require more than one person to be done, we could not find any research comparing individual effectiveness with pair or team effectiveness.

Working in teams can boost motivation [25] and promote the use of each member's strengths [26] (e.g., some CST clients complain that test reports are poorly written [27]; therefore, having one person on each team with good writing skills could improve the reports).

CST has challenges related to the high number of duplicated test reports [28,29] and the high false-positive rate for identifying defects [10,18,30]. If a crowd is organized into teams, it is possible to mitigate these challenges because a smaller number of reports is produced (i.e., there are fewer duplicated test reports) and because more eyes review and evaluate the claimed defects (i.e., there is a lower false-positive rate).

However, the anticipated advantages of team-based CST do not mean that it comes without challenges. In fact, teamwork requires extra effort because of the need to coordinate tasks among team members [31]. If time is limited for software testing, this may reduce the effectiveness of the testing. Additionally, teamwork requires a certain homogeneity among team members to facilitate communication and enhance harmony [32].

### 3. Approach

#### 3.1. Research Aim and Questions

The main aim of this research was to study the impact of a coordinated crowd on the effectiveness and efficiency of exploratory functional testing. We asked two research questions in this study:

- (1) What is the effect of using *collaborative CST*, in which pairs of collaborating workers identify defects, as compared with *conventional CST*, which uses non-collaborating crowd workers?
- (2) Is there any difference with respect to the level of difficulty of the defects detected between these two approaches of CST?

The first question was concerned with the impact of using collaborating pairs in terms of the total number of defects found for a particular software application and the total number of invalid defects submitted. The invalid defects are the false defects reported by workers in addition to the defects that are identified more than once, which we will call duplicate defects. Each collaborating pair does not include duplicate defects in their reports, but non-collaborating individuals do because they are merging the results of different test reports. This situation is the reality in the domain of CST. Reports must be checked to see whether the same defects are appearing multiple times.

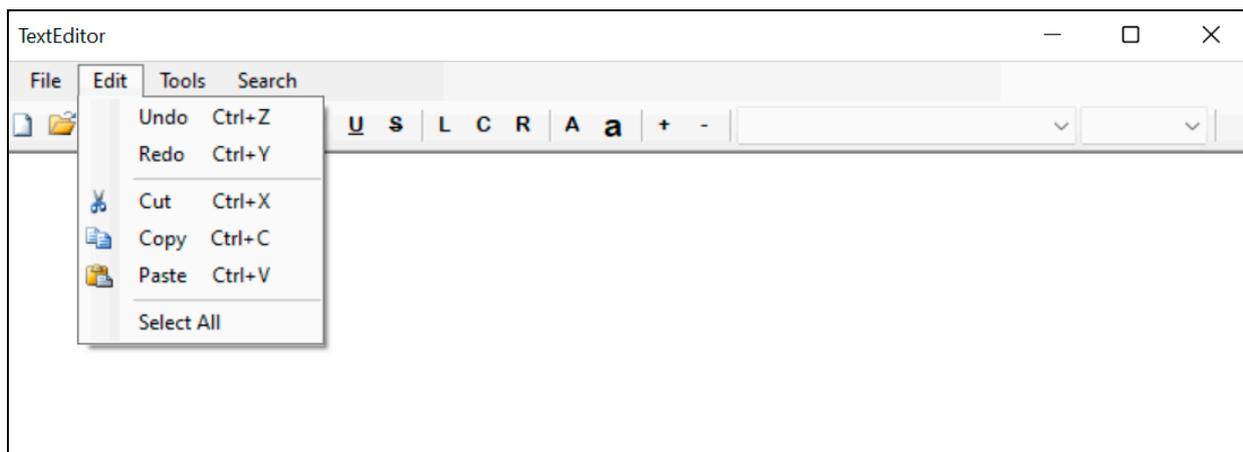
The second question was concerned with the effects of having two workers communicate with each other on the possibility of detecting more difficult defects. The motivation for this question was that microtasking (i.e., using a set of individual workers) had been criticized for its inferior results in detecting complex defects [33–35]. We wanted to see whether this was true and, if so, whether working in pairs could mitigate this potential challenge in the domain of software testing.

#### 3.2. Subjects and Experimental Setup

This research was conducted by final-year students in a Bachelor of Science program who had a good background in software testing. They were in a Software Engineering course in the Information Systems Department, King Saud University, Saudi Arabia. A

total of 75 students were involved in the experiment. They were divided on the basis of their performance into 25 “groups” of one person and 25 groups of two persons.

It is not uncommon to use student subjects in software engineering research [36]. Bitchy [37] required that they be trained sufficiently well to perform the experiment. In this research, students had been taught testing techniques and methods in a previous course, and this was the second course that included software testing. Additionally, they had been given a short introduction on the software to be tested and informed about the experiment’s instructions (e.g., shown the test report form, told how much time would be needed for the research). The software application being tested was a simple text editor developed in C# purposely for this research. The main screen is shown in Figure 1. A text editor was used for this research for the sake of simplicity. It included basic functionalities that are commonly found in several applications the students use, such as file list functions, including New, Open, Save, and Exit; edit list functions, including Undo, Redo, Copy, Cut, Paste, and Select All; and customizing and search functions, along with the standard text editing functions shown in Figure 1 (e.g., changing the font size and type, text alignment). The application was seeded with a total of 44 defects. The full description of the defects, the test reports submitted by the workers, and the software being tested can be found with this link (<https://github.com/salyahya99/Collaborative-CST>, accessed on 1 September 2022). In this research, we focused mainly on the functional testing type, so all the defects were functional. We also injected defects with various difficulty levels to test whether the effectiveness of detection changed according to the difficulty level.



**Figure 1.** Screenshot of the main screen with the “Edit” menu shown.

The experiment was a course assignment for the students, and thus, it was expected that they do their best when performing the testing. Individual groups and collaborative groups were asked to conduct the experiment for a fixed-length testing session of 2 h. They were asked to record the testing session. The collaborative groups were asked to utilize the testing session for both conducting the tests and reporting the results. They were encouraged to share their thoughts with each other and mutually evaluate their work. In order to simulate the CST environment, they communicated with each other via the Internet. Most of them used Zoom software to do so.

Before starting the testing, the application was introduced to the students with a brief description of its main functionalities, and the testing form was explained in detail. The testing form was a standard form that testers commonly use in software testing. The main components of the form included brief descriptions of the defect, inputs (illustrating the input procedure/data that caused the defect to happen), expected results (illustrating the ideal output of the tester after the test was performed), and actual results (illustrating the actual output after the test was performed).

### 3.3. Data Analysis

Each group was asked to submit one test report. The two workers in each collaborative group performed the testing together but produced one test report. After the test reports had been completed, they were evaluated by the researcher and an external senior tester who had 12 years of experience in the software industry in general and 5 years in software testing. A total of 687 submitted defects were reviewed. Each submitted defect was evaluated as to whether it was true or false and mapped to one of the designated defects identified before starting the experiment. There were only a few differences between the two evaluations (nine reported defects), and they were due to vague information on some of the reported defects. For example, some defects were described correctly in the defect description field but the inputs used did not lead to the reported actual results. The validators initially marked them and later worked together to come up with a final decision about them. The mapping was documented in the same test reports in a new column titled Designated Defect ID. For each true defect reported, we determined the defect number to which it corresponded in our list.

This research focused on comparing the performance between two non-collaborating workers (NCW) and two collaborating workers (CW). The performance of the NCWs was calculated by considering the unique valid defects identified in addition to the false and duplicate defects identified in the combinations of any two single workers. Since we had 25 single workers in the NCW approach, we developed a small program in Python to calculate the defect data of 300 combinations, which would address all the possible cases of forming pairs (i.e., [1,2], [1,3], ..., [2,3], [2,4], ..., [24,25]).

The effect of the CWs was measured by identifying the mean difference between the CWs and the NCWs. In software engineering research, it is recommended that the unstandardized effect such as the mean difference for assessing practical importance be reported [38]. Additionally, Wilkinson [39] advised that if the units of measurement were meaningful on a practical level (e.g., number of defects in this research), then it was usually preferred to use an unstandardized measure. For all of the results, we also reported the confidence interval (CI) and measured the degree of certainty using a confidence level of 95%.

We computed the invalid defects resulting from false-positive defects only and the invalid defects resulting from the total sum of the false-positive defects plus the duplicate defects. The reason for this differentiation was that we expect researchers to start helping with detecting duplicate defects through using some artificial intelligence techniques in the near future. Although there had not been much improvement in this area (only one study [29] identified by an SLR paper published recently [7]), and we were unaware of any CST platform that incorporated these techniques, we believed it was worth making the results more detailed by reporting both sources of invalid defects, the false-positive defects and the duplicate defects.

### 3.4. Measures

This research focused on comparing the performances of the NCW and CW approaches with regard to the number of valid defects found, the number of invalid defects reported, and the difficulty of the defects detected. For each of these areas, we used measures that are commonly adopted in the software engineering community [14].

For the valid defects, we computed the average number of unique valid defects identified by each two NCWs and each two CWs. We then used this information to compute the detection effectiveness. The detection effectiveness is defined as the ratio of the average number of unique valid defects identified to the total defects available for detection (44 defects in this research). It is represented mathematically as follows:

$$E = \frac{vd}{td}$$

where  $E$  is the detection effectiveness,  $vd$  is the average number of unique valid defects identified, and  $td$  is the total defects available for detection.

With regard to the invalid defects, we computed the average number of invalid defects reported by the NCW and CW groups and then computed the validity. The validity is defined as the ratio of the number of valid defects detected to the total number of valid defects plus invalid defects detected. The mathematical representation of validity is as follows:

$$V = \frac{vd}{vd + id}$$

where  $V$  is the validity,  $vd$  is the average number of unique valid defects identified, and  $id$  is the average number of invalid defects reported.

Finally, we computed the average level of difficulty of defect detection per pair for both the NCWs and CWs. The defects were classified based on the difficulty of detection using the measure applied by Itkonen et al. [40]. The measure was derived from the definition of the failure-triggering fault interaction (FTFI) number [41], which is the number of conditions required to trigger a failure. It is determined by studying how many different inputs or actions are required to make the failure occur. Four levels of difficulty of defect detection were used, mode-0 to mode-3. A mode-0 defect was a defect that was immediately apparent to the tester (e.g., a redundant function), and a mode-3 defect required three inputs/conditions or more to be detected.

#### 4. Findings

This section presents the findings of this research regarding valid defects, invalid defects, and defect detection difficulty.

##### 4.1. Valid Defects

Table 1 illustrates the average number of unique valid defects identified by two NCWs and two CWs. As is clear from the table, the approaches of the NCWs and CWs achieved almost the same accuracy of detection; there is a slight difference in favor of the NCW approach. The NCWs detected 14.81 defects, with a CI of 14.4 to 15.2, and the CWs detected 14.48 defects, with a CI of 12.5 to 16.5. There was an average of 0.33 more defects (2.22%) detected by the NCWs.

**Table 1.** The average number of unique valid defects and the effectiveness of detection for NCWs and CWs.

Approach	Average Number of Unique Valid Defects	Detection Effectiveness
NCW	14.81	33.66%
CW	14.48	32.91%

We also computed the detection effectiveness (see Section 3.4 for definition). The results show that the NCWs were slightly more effective (33.66%) than the CWs (32.91%).

##### 4.2. Invalid Defects

Table 2 presents the average number of invalid defects reported by the NCW and CW groups. As defined in Section 3.1, invalid defects detected by CWs are always false defects. However, invalid defects detected by NCWs are false defects plus duplicate defects. Table 2 gives details of the invalid defect data by presenting the false defects and duplicate defects separately.

**Table 2.** Invalid defect findings.

Approach	Average Unique Valid Defects	Average False Defects	Average Duplicate Defects	Total Invalid Defects	Average Submitted Defects	Validity (False Defects Only)	Validity (Total Invalid Defects)
NCW	14.81	2.88	5.91	8.79	23.6	83.72%	62.75%
CW	14.48	1.20	-	1.20	15.68	92.35%	92.35%

The results show that the CWs significantly outperformed the NCWs. The NCWs detected an average of 8.79 invalid defects (CI 8.48 to 9.1), whereas the CWs detected an average of 1.20 invalid defects (CI 0.681 to 1.72). This means that there was an 86.35% reduction in invalid defects reported by workers using the CW approach. This was a significant improvement in the inaccurate detection of many invalid defects in CST.

When ignoring the duplicate defects and focusing on measuring the difference in false defects only, we still noticed that the CWs reported fewer false defects (1.20 defects, as mentioned earlier) than the NCWs (2.88 defects), with a CI of 2.67 to 3.09. This was a 58.33% reduction in reported false defects.

We also computed the validity of both approaches (see Section 3.4 for definition). The results show that the validity of the CW detections (92.35%) was significantly higher than that of the NCW detections (62.75%). If we subtract the duplicate defects from the total detections of the NCWs, the groups' validity improves to 83.71%; it is still evident, however, that the CWs had considerable improvement in validity.

#### 4.3. Defect Detection Difficulty

Table 3 shows the average level of difficulty of defect detection per pair for NCWs and CWs. As illustrated in Section 3.2, we classified the detection difficulty as having four levels (0 to 3). The results showed that the CWs succeeded in detecting more difficult defects (both level-2 and level-3 defects). Each collaborating pair detected an average of 3.2 level-2 defects (CI 2.55 to 3.85) and 1.16 level-3 defects (CI 0.621 to 1.7), while each non-collaborating pair discovered an average of 2.68 level-2 defects (CI 2.56 to 2.8) and 0.92 level-3 defects (CI 0.811 to 1.03).

**Table 3.** Findings on defect detection difficulty.

Type of Workers	Level 0 (Easiest)	Level 1	Level 2	Level 3 (Hardest)
NCW	1.8	9.41	2.68	0.92
CW	1.96	8.16	3.2	1.16
Improvement	8.89%	−13.28%	19.40%	26.09%
	−9.72%		21.22%	

The average percentage of improvement of the CWs over the NCWs for level-2 defects was 19.40% and for level-3 defects was 26.09%. If we compute the overall improvement for defect levels 2 and 3, we can conclude that there was a 21.22% improvement in detecting difficult defects.

The detection of level-0 defects had similar results; the NCW approach detected 1.8 defects, with a CI of 1.74 to 1.86, and the CW approach detected 1.96 defects, with a CI of 1.49 to 2.43. The detection of level-1 defects by the NCWs was 9.41, with a CI of 9.18 to 9.64, which exceeded the detection by the CWs, at 8.16, with a CI of 7.21 to 9.11.

There was only a slight improvement in the detection of level-0 defects by the CWs over the NCWs (8.89%) and a slightly negative effect for level-1 defects (−13.28%). The overall result for the detection of the simple defects (defect level-0 and level-1) favored the conventional approach (NCWs). The CWs reduced the overall percentage of detected simple defects by 9.72%. A detailed view of the improvement results is shown in Figure 2.



**Figure 2.** Improved percentage of detection by CWs of each level of defect difficulty.

## 5. Discussion

The findings show that the non-collaborative testers had a slightly better ability to detect more valid defects than the collaborative testers. It was expected that individual workers would detect more defects because they would not need to spend some of their time communicating with each other as collaborative CST workers would. However, a main reason that a larger gap did not occur between the two approaches was because many of the defects detected by the NCW pairs were identified by both individuals, leading to a smaller number of unique defects detected and resulting in a waste of effort. On the contrary, although collaboration by pairs added some “overhead” due to the need for coordination in a joint activity, this did not significantly impact the effectiveness of defect detection. However, we believe that increasing the number of CWs on one team (i.e., having three workers or more) would influence the effectiveness of the team remarkably since more time would be needed for coordination among the workers.

Regarding the invalid defects, we noticed that there was a major improvement in detection when the collaborative approach was applied. The improvement originated from two sources. First, the collaborating pairs did not report duplicated defects because they worked together to produce one test report. The average percentage of duplicated defects was approximately 25% of the overall number of defects submitted by each non-collaborating pair. Second, the experiment showed that working as collaborating pairs reduced the average number of submitted false defects by almost 60%.

The question is what could have contributed to this significant cut in false defects? We believe that a main reason for this was that the pairs analyzed the claimed defects better when working together. They shared their thoughts with each other and received critiques and evaluations during an open and continuous discussion in the testing session. This likely helped to produce better-quality test reports than those of the non-collaborating individuals, who were working in isolation. The power of teamwork was a possible reason for the boost in the validity of collaborating pairs; it could enhance the workers’ motivation and make the testing task more enjoyable and less stressful. Advancements in the science of psychology support the argument that even though some people may think the effect of teamwork is minor or even unfavorable, teamwork can actually make the work of its members more effective than the work of individual workers [42].

The collaborating pairs were also able to outperform the non-collaborating pairs in detecting the difficult defects, with a roughly 21% improvement. It is assumed here that spending time in conversation and mutual evaluation helped workers reach a deep level of

analysis of the functions of the system being tested, and this in turn resulted in detecting defects that did not appear on the surface. Since there was a fixed time to complete the testing session, we think that this improvement came at the expense of the number of simple defects found, although the number was only slightly lower, at almost  $-10\%$ .

The findings showed that working as collaborating pairs can be as effective in detecting defects as working individually; any differences were marginal. However, collaborative software testing significantly affects the quality of test reports submitted in two ways: it helps reduce the invalid defects and also helps detect the more difficult defects. These two are considered main problems in CST activities [4–7]. From a practical view, the findings suggest that CST platforms can benefit from developing new mechanisms that allow the formation of teams of two individuals who can participate in testing jobs. Depending on the importance of the system under testing, it would be useful to at least nominate a combination of individual workers and CWs.

It is important to mention that this research did not consider the process of forming tester pairs. The pairs were students who knew each other, and most of them had studied together in the last 3 years. This homogeneity was less likely to cause an issue in the experiment's setting. If the use of collaborating pairs were to be applied in CST, it would be necessary to think about how to build homogeneous teams to facilitate communication and enhance harmony [32]. Crowdsourcing by nature consists of heterogeneous individuals who live in different time zones, belong to various cultures, and have diverse levels of skills. Although we did not find research in the literature about forming teams in CST, many models have been suggested to improve homogeneity in the generic domain of crowdsourcing, such as measuring the willingness to collaborate [43], the worker-to-worker affinity [24], the social context recommendation [44], and the worker-to-worker rating [45]. Additionally, the author's previous experience with some crowdsourcing platforms such as Upwork showed that it would be useful to let individuals look for other persons themselves and then start working on testing tasks together. This approach can promote the possibility of finding suitable pairs.

Furthermore, the study was concerned about exploratory functional testing only. We do not know if it is possible to generalize the findings to other types of testing (i.e., nonfunctional testing such as usability testing or performance testing). Our findings suggested that when a testing task requires critical thinking and deep analysis, then it is likely to benefit from collaborative testing. On the contrary, the microtasking model can work well for testing jobs that are divisible, which are normally of low complexity. As an example of a case that does not seem appropriate for collaborative testing, we offer the following: many software houses use a crowd to make sure the performance of their built system is acceptable when used by different devices or in different locations around the world. They require merely a crowd who uses specific types of hardware or lives in certain countries to run the system. In these situations, collaborative testing would not add tangible value.

## 6. Conclusions

This research examined the effect of using pairs of CWs in CST and how it would differ from the conventional approach that uses NCWs. The findings are promising since they show that collaborative teams can produce reports of significantly better quality; they report fewer invalid defects and identify more serious valid defects. The improvement in quality did not affect the quantity of the valid defects detected; there was only a marginal effect.

There are several possible directions in which this research could proceed. First, it would be interesting to think of ways to form pairs in CST. The formation models and approaches mentioned in Section 5 could be a starting point for proposing appropriate mechanisms built especially for CST activities. Moreover, the collaborative approach used in this research could be evaluated not only with other types of software testing but also with other software engineering activities. There is an increasing interest in using

crowdsourcing for requirements engineering and software design. However, the current research studies and practices in these activities still focus on conventional microtasking.

**Funding:** This research received no external funding.

**Data Availability Statement:** Not Applicable.

**Conflicts of Interest:** The author declares no conflict of interest.

## References

1. Alyahya, S.; Alsayyari, M. Towards better crowdsourced software testing process. *Int. J. Coop. Inf. Syst.* **2020**, *29*, 2040009. [CrossRef]
2. Alyahya, S.; Alrugebh, D. Process Improvements for Crowdsourced Software Testing. *Int. J. Adv. Comput. Sci. Appl.* **2017**, *8*, 32–40. [CrossRef]
3. LaToza, T.D.; Van Der Hoek, A. Crowdsourcing in software engineering: Models, motivations, and challenges. *IEEE Softw.* **2016**, *33*, 74–80. [CrossRef]
4. Chen, X.; Jiang, H.; Chen, Z.; He, T.; Nie, L. Automatic test report augmentation to assist crowdsourced testing. *Front. Comput. Sci.* **2019**, *13*, 943–959. [CrossRef]
5. Winkler, D.; Sabou, M.; Petrovic, S.; Carneiro, G.; Kalinowski, M.; Biffel, S. Improving model inspection processes with crowd-sourcing: Findings from a controlled experiment. *Commun. Comput. Inf. Sci.* **2017**, *748*, 125–137. [CrossRef]
6. Jiang, H.; Chen, X.; He, T.; Chen, Z.; Li, X. Fuzzy Clustering of Crowdsourced Test Reports for Apps. *ACM Trans. Internet Technol.* **2018**, *18*, 1–28. [CrossRef]
7. Alyahya, S. Crowdsourced software testing: A systematic literature review. *Inf. Softw. Technol.* **2020**, *127*, 106363. [CrossRef]
8. Musson, R.; Richards, J.; Fisher, D.; Bird, C.; Bussone, B.; Ganguly, S. Leveraging the crowd: How 48,000 users helped improve Lync performance. *IEEE Softw.* **2013**, *30*, 38–45. [CrossRef]
9. Malone, D.; Dunne, J. Social dogfood: A Framework to Minimise Clouc Field Defects through crowd Sourced TESTING. In Proceedings of the 2017 28th Irish Signals and Systems Conference, Killarney, Ireland, 20–21 June 2017. [CrossRef]
10. Liu, D.; Bias, R.G.; Lease, M.; Kuipers, R. Crowdsourcing for usability testing. *Proc. ASIST Annu. Meet.* **2012**, *49*, 1–10. [CrossRef]
11. Wu, G.; Cao, Y.; Chen, W.; Wei, J.; Zhong, H.; Huang, T. AppCheck: A Crowdsourced Testing Service for Android Applications. In Proceedings of the 2017 IEEE 24th International Conference on Web Services, Honolulu, HI, USA, 25–30 June 2017. [CrossRef]
12. Raymond, E. The cathedral and the bazaar. *Knowl. Technol. Policy* **1999**, *12*, 23–49. [CrossRef]
13. Steiner, I.D. *Group Process and Productivity*; Academic Press: Cambridge, MA, USA, 1972.
14. Mäntylä, M.V.; Itkonen, J. More testers—The effect of crowd size and time restriction in software testing. *Inf. Softw. Technol.* **2013**, *55*, 986–1003. [CrossRef]
15. Fry, Z.P.; Weimer, W. A human Study of Fault Localization Accuracy. In Proceedings of the IEEE International Conference on Software Maintenance, Timisoara, Romania, 12–18 September 2010. [CrossRef]
16. Nielsen, J. Finding Usability Problems through Heuristic Evaluation. In Proceedings of the Conference on Human Factors in Computing Systems Proceedings, New York, NY, USA, 1 June 1992. [CrossRef]
17. Zhao, Y.; Feng, Y.; Wang, Y.; Hao, R.; Fang, C.; Chen, Z. Quality assessment of crowdsourced test cases. *Sci. China Inf. Sci.* **2020**, *63*, 190102. [CrossRef]
18. Zogaj, S.; Bretschneider, U.; Leimeister, J.M. Managing crowdsourced software testing: A case study based insight on the challenges of a crowdsourcing intermediary. *J. Bus. Econ.* **2014**, *84*, 375–405. [CrossRef]
19. Kit, E.; Finzi, S. *Software Testing in the Real World: Improving the Process*; ACM Press/Addison-Wesley Publishing Co.: Boston, MA, USA, 1995.
20. Research Triangle Institute. The Economic Impacts of Inadequate Infrastructure for Software Testing. Planning Report 02-3. 2002. Available online: [https://lara.epfl.ch/w/\\_media/misc/rti02economicimpactsinadequateinfrastructuresoftwaretesting.pdf](https://lara.epfl.ch/w/_media/misc/rti02economicimpactsinadequateinfrastructuresoftwaretesting.pdf) (accessed on 1 September 2022).
21. Naik, K.; Tripathy, P. *Software Testing and Quality Assurance: Theory and Practice*; John Wiley & Sons: New York, NY, USA, 2008.
22. Goodpasture, J.C. Project Management the Agile Way: Making It Work in the Enterprise. In *Teams Are Everything—Chap 8*; J. Ross Publishing: Fort Lauderdale, FL, USA, 2010.
23. Rahman, H.; Roy, S.B.; Thirumuruganathan, S.; Amer-Yahia, S.; Das, G. Task Assignment Optimization in Collaborative Crowdsourcing. In Proceedings of the IEEE International Conference on Data Mining, ICDM, Atlantic City, NJ, USA, 14–17 November 2015; Volume 2016-January. [CrossRef]
24. Ikeda, K.; Morishima, A.; Rahman, H.; Roy, S.B.; Thirumuruganathan, S.; Amer-Yahia, S.; Das, G. Collaborative Crowdsourcing with Crowd4U. *Proc. VLDB Endow.* **2016**, *9*, 1497–1500. [CrossRef]
25. Rokicki, M.; Zerr, S.; Siersdorfer, S. Groupsourcing: Team competition designs for crowdsourcing. In Proceedings of the WWW 2015 the 24th International Conference on World Wide Web, Florence, Italy, 18–25 May 2015.
26. Black, R. *Managing the Testing Process*; John Wiley & Sons: New York, NY, USA, 2009.

27. Liu, D.; Zhang, X.; Feng, Y.; Jones, J.A. Generating Descriptions for Screenshots to Assist Crowdsourced Testing. In Proceedings of the 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, 20–23 March 2018; Volume 2018-March, pp. 492–496.
28. Gao, R.; Wang, Y.; Feng, Y.; Chen, Z.; Eric Wong, W. Successes, challenges, and rethinking—An industrial investigation on crowdsourced mobile application testing. *Empir. Softw. Eng.* **2019**, *24*, 537–561. [[CrossRef](#)]
29. Wang, J.; Li, M.; Wang, S.; Menzies, T.; Wang, Q. Images don't lie: Duplicate crowdtesting reports detection with screenshot information. *Inf. Softw. Technol.* **2019**, *110*, 139–155. [[CrossRef](#)]
30. Winkler, D.; Sabou, M.; Petrovic, S.; Carneiro, G.; Kalinowski, M.; Biffi, S. Improving Model Inspection with Crowdsourcing. In Proceedings of the 4th International Workshop on CrowdSourcing in Software Engineering, Buenos Aires, Argentina, 22 May 2017; pp. 30–34.
31. Malone, T.W.; Crowston, K. The interdisciplinary Study of Coordination. *ACM Comput. Surv.* **1994**, *26*, 87–119. [[CrossRef](#)]
32. Qamar, N.; Malik, A.A. Birds of a Feather Gel Together: Impact of Team Homogeneity on Software Quality and Team Productivity. *IEEE Access* **2019**, *7*, 96827–96840. [[CrossRef](#)]
33. Novak, J. Collective action and human computation: From crowd-workers to social collectives. In *Handbook of Human Computation*; Springer: New York, NY, USA, 2013.
34. Kim, S.; Robert, L.P. Crowdsourcing Coordination: A Review and Research Agenda for Crowdsourcing Coordination Used for Macro-tasks. *Macrotask Crowdsource.* **2019**, *2019*, 17–43.
35. Blohm, I.; Zogaj, S.; Bretschneider, U.; Leimeister, J.M. How to manage crowdsourcing platforms effectively? *Calif. Manag. Rev.* **2018**, *60*, 122–149. [[CrossRef](#)]
36. Kitchenham, B.A.; Pfleeger, S.L.; Pickard, L.M.; Jones, P.W.; Hoaglin, D.C.; El Emam, K.; Rosenberg, J. Preliminary guidelines for empirical research in software engineering. *IEEE Trans. Softw. Eng.* **2002**, *28*, 721–734. [[CrossRef](#)]
37. Tichy, W.F. Hints for reviewing empirical work in software engineering. *Empir. Softw. Eng.* **2000**, *5*, 309–312. [[CrossRef](#)]
38. Kampenes, V.B.; Dybå, T.; Hannay, J.E.; Sjøberg, D.I.K. A systematic review of effect size in software engineering experiments. *Inf. Softw. Technol.* **2006**, *48*, 745–755. [[CrossRef](#)]
39. Wilkinson, L. Statistical methods in psychology journals: Guidelines and explanations. *Am. Psychol.* **1999**, *54*, 594–604. [[CrossRef](#)]
40. Itkonen, J.; Mäntylä, M.V.; Lassenius, C. Defect Detection Efficiency: Test Case Based vs. Exploratory Testing. In Proceedings of the 1st International Symposium on Empirical Software Engineering and Measurement, Madrid, Spain, 20–21 September 2007. [[CrossRef](#)]
41. Kuhn, D.R.; Wallace, D.R.; Gallo, A.M. Software fault interactions and implications for software testing. *IEEE Trans. Softw. Eng.* **2004**, *30*, 418–421. [[CrossRef](#)]
42. Salas, E.; Reyes, D.L.; McDaniel, S.H. The science of teamwork: Progress, reflections, and the road ahead. *Am. Psychol.* **2018**, *73*, 714–723. [[CrossRef](#)] [[PubMed](#)]
43. Ye, L.; Wang, X.; Sun, H.; Wang, J. Personalized Teammate Recommendation for Crowdsourced Software Developers. In Proceedings of the ASE 2018—Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, New York, NY, USA, 3–7 September 2018. [[CrossRef](#)]
44. Lee, D.H.; Brusilovsky, P.; Schleyer, T. Recommending collaborators using social features and MeSH terms. *Proc. Proc. ASIST Annu. Meet.* **2011**, *48*, 1–10. [[CrossRef](#)]
45. Lykourantzou, I.; Kraut, R.E.; Wang, S.; Dow, S.P. Team Dating: A Self-Organized Team Formation Strategy for Collaborative Crowdsourcing. In Proceedings of the Conference on Human Factors in Computing Systems, New York, NY, USA, 7–12 May 2016. [[CrossRef](#)]