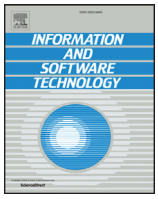
Information and Software Technology 127 (2020) 106363

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Crowdsourced software testing: A systematic literature review

Sultan Alyahya

Information Systems Department, King Saud University, Saudi Arabia

ARTICLE INFO

Keywords:
Crowdsourcing
Software testing
Crowdsourced software testing
Empirical software engineering
Systematic literature review

ABSTRACT

Context: Crowdsourced software testing (CST) refers to the use of crowdsourcing techniques in the domain of software testing. CST is an emerging area with its applications rapidly increasing in the last decade.

Objective: A comprehensive review on CST has been conducted to determine the current studies aiming to improve and assess the value of using CST as well as the challenges identified by these evaluation studies.

Method: We conducted a systematic literature review on CST by searching six popular databases. We identified 50 primary studies that passed our quality assessment criteria and defined two research questions covering the aim of the study.

Results: There are three main process activities that the current literature aims to improve, namely selection of suitable testers, reporting of defects, and validation of submitted defects. In addition, there are 23 CST evaluation studies and most of them involve a large group and real crowd. These studies have identified 27 different challenges encountered during the application of crowdsourcing in software testing.

Conclusions: The improvements achieved for the specific process activities in CST help explore other unexplored process activities. Similarly, knowing the characteristics of the evaluation studies can direct us on what other studies are worth investigating. Additionally, many of the challenges identified by the evaluation studies represent research problems that need better understanding and alternative solutions. This research also offers opportunities for practitioners to understand and apply new solutions proposed in the literature and the variations between them. Moreover, it provides awareness to the related parties regarding the challenges reported in the literature, which they may encounter during CST tasks.

Contents

E-mail address: sualyahya@ksu.edu.sa

https://doi.org/10.1016/j.infsof.2020.106363
Received 27 August 2019; Received in revised form 28 May 2020; Accepted 30 May 2020
Available online 8 June 2020

1. Introduction

Crowdsourced software testing (CST) refers to the use of crowdsourcing techniques in the field of software testing [1]. It is an emerging area with rapidly increasing use in the last decade. It has been applied to many software testing activities such as usability testing [2], functional testing [3], compatibility testing [4], and performance testing [5].

Many software platforms, such as uTest[1] and CrowdFlower[2] (recently named FigureEight), have appeared to support the application of this new technique. These platforms enable requesters (e.g. software organisations that require help in testing their products) to submit their testing requirements to a large pool of workers, who perform testing tasks and submit test reports with relatively lower compensations [6]. The appearance of the platforms has multiplied CST activities and consequently has drawn the attention of the academic community to explore CST to add new improvements and make reliable assessments to advance this new technique of software testing.

The term "crowd" traditionally refers to a large and undefined group of heterogeneous individuals who perform a crowdsourced task [7]. Nevertheless, for privacy reasons (e.g. privacy concerns), a private crowd working in software engineering activities, such as software testing, can also be defined, consisting of employees working within the organization that requests the task [8].

CST is required in various scenarios, for example, when a software company lacks testers specialized in a specific testing type such as security testing. This type of testing may be challenging, even for expert internal testing teams. Conventional approaches may fail to cover all security gaps due to the evolving tactics of hackers. In addition, usability testing is best conducted by end users. Hence, crowdsourcing usability testing can benefit from end users' representations and domain expertise recommendations [1]. For example, a teacher using a learning application is likely to provide more realistic usability feedback than a software tester with no teaching experience. Moreover, a pool of testers can help run compatibility testing. It would be challenging to conduct full compatibility testing in-house, given the large number of evolved devices and their operating systems as well as the time and cost constraints. When devices are outdated, the overall return on the investment made for procuring such devices for testing purposes decreases. Therefore, using a crowd of testers, end users, or domain experts who own such devices and can freely use them for short-term testing may be helpful.

Although the CST work has widely adopted several types of software testing and applied different testing contexts, it lacks a proper understanding of how the individual studies in CST contributes to the entire field of CST. Leicht [9] also observed that CST research lacks proper 'terminology' and classification with respect to existing software testing practices.

Therefore, this research aims to fill this gap by presenting a comprehensive review of current research progress in the field of CST, which can indicate the maturity of the field and help understand how further research can be conducted. In particular, a systematic literature review (SLR) is provided with two-fold purpose. First, we would like to understand what are the current studies aiming to improve CST. Second, we aim to investigate what are the current studies assessing the value of using CST and determine what challenges have been identified by these evaluation studies.

The remainder of this paper is divided as follows: Section 2 describes the related work, Section 3 presents the research methodology, Section 4 reports the findings, Section 5 provides the discussion of the findings, and finally, Section 6 concludes the study.

2. Related work

Testing is an essential activity in software engineering. Quality, in terms of validity and verifiability, determines the success of a software project. Many current software products are used by thousands or even millions of people, helping them do their jobs effectively and efficiently or, alternately, causing them untold frustration and the costs of lost work or lost business [10].

Testing is defined as "the process of executing a program with the intent of finding errors" [10]. Hence, it is a destructive process of trying to find errors whose presence is assumed in a program. Its main goal is to establish a certain confidence that a program does what it is supposed to do. However, software testing cannot guarantee the complete absence of errors; instead, software testing attempts to be as complete as possible by identifying the largest possible number of errors [10]. Sterlings [11] observed that "the best software is that which has been tested by thousands of users under thousands of different conditions". This is where the concept of crowdsourcing can help.

Crowdsourcing refers to the practice of using the Internet to outsource work to the crowd [7]. It is a type of participative online activity in which an individuals or organisations propose to a group of people of varying knowledge and heterogeneity, via an open call, the voluntary undertaking of a task [12]. The main benefits of using crowdsourcing include its highly collaborative environment, its suitability to examine ideas, its speedy process, and its relatively low cost [13].

As the above definition suggests, crowdsourcing is a model that relies on using information technology capabilities (e.g., the Internet) as the main medium between requesters and crowd. It can help provide solutions to problems or perform tasks by broadcasting the need through online communities. The outcome of the crowd's work then becomes the property of the problem or task requesters. Normally, crowd members are rewarded for their work with monetary incentives or by other means, such as social appreciation.

Crowdsourcing techniques have been applied to software development activities such as requirements engineering [14], design [15], implementation [16], and testing [17]. Software crowdsourcing adopts an adaptive, open, and distributed development process model. Internet-based software platforms offer online communities and market functionalities that facilitate and coordinate thousands of distributed community members in the construction and evolution of large-scale software systems. Famous successful software crowdsourcing platforms include uTest and TopCoder.[3]

CST is an emerging trend in software engineering. Companies and developers outsource testing tasks to thousands of online workers who are thus provided with more job opportunities and extra income, and requesters are provided with fast, high-quality services at low cost [1].

[1] http://www.utest.com/
[2] http://www.figure-eight.com/
[3] http://www.topcoder.com/

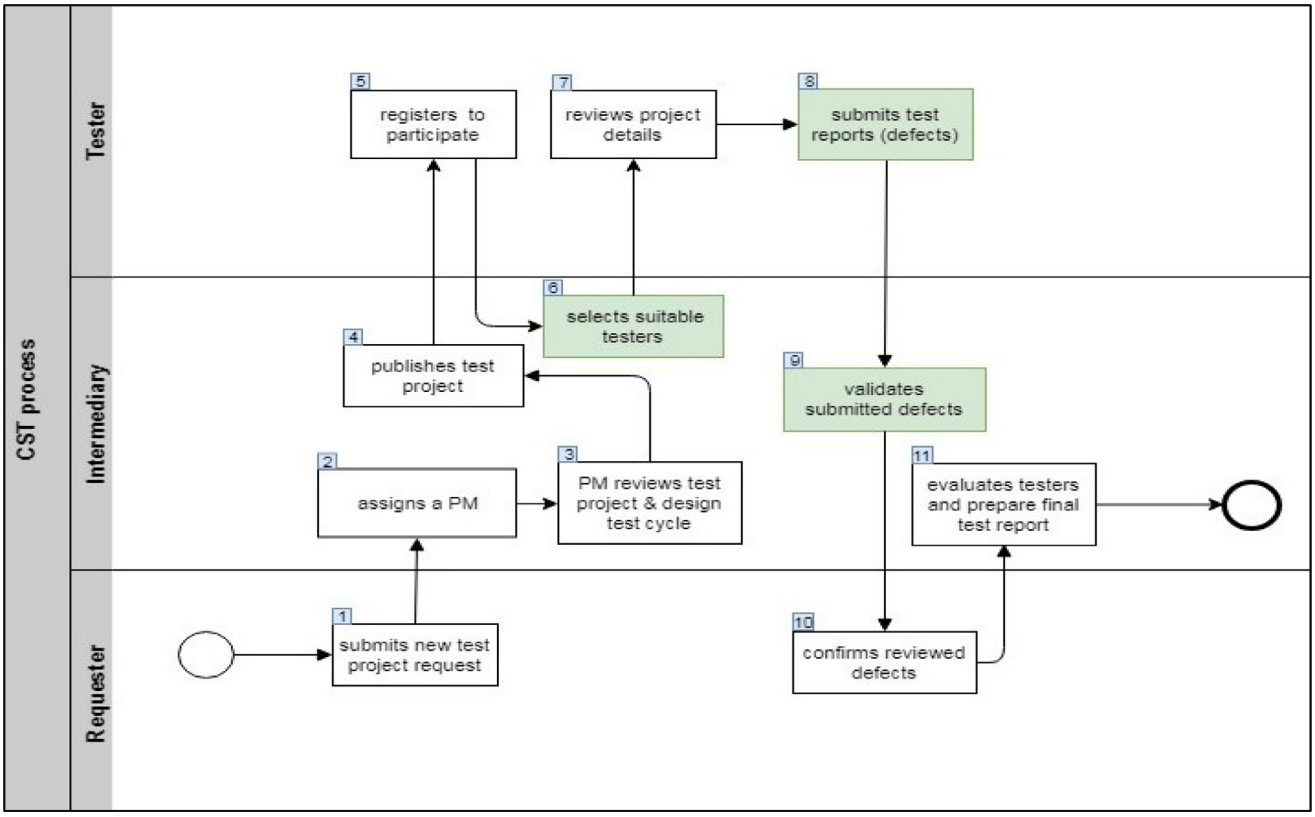


Fig. 1. CST process (highlighted activities will be discussed later in Section 4.1).

It has been widely recognised that CST has three main components [18–21]:

- Requester: the person or company that needs to test a product (also known as owner).
- Tester: online workers aiming to utilise their time and gain some benefits (e.g. improving testing skills, financial).
- CST platform: an online portal with testing services that work as an intermediary between requesters and testers.

Fig. 1 shows a graphical representation of a typical CST process [6,1,22]. The activities in the process are numbered from 1 to 11. The process starts when a software company (i.e., the requester in this case) shares its testing requirements with a CST service provider (Activity 1). Testing requirements may include the type of testing and the types/configurations of devices that the system should be tested on. For example, the requester may ask the crowd to test a website. He may need to get usability testing (i.e. how usable the system is to end-users). In this scenario, the requester may have difficulty in finding locally representative people to give feedback about the website's usability. Hence, CST platforms can provide an alternative solution.

A CST platform coordinator nominates a project manager (PM) based on test requirements and availability (Activity 2). A PM is normally a community leader or an employee in the CST company. With the help of the project manager, a test plan is designed (Activity 3) and shared with the testers registered on the platform (Activity 4). The plan may include dividing the website testing job into a set of tasks where each task involves testing a user scenario (i.e., use case). Testers create their profiles by providing all the necessary information such as their qualifications, experiences, skills, location, availability, and devices. They can then check the CST dashboard for a list of open projects and register their interest to participate (Activity 5). The PM and the requester nominate a set of testers based on their profiles and possibly their testing histories (Activity 6). For the usability testing example, testers can be ordinary end-users from both genders and diverse age groups and education levels. The PM and requester then wait for the testers to review and perform project tasks (Activity 7) and submit testing reports (Activity 8), which they then evaluate. In most CST platforms, the PM selects the most interesting bugs until the agreed-upon number of bugs is reached (Activity 9), and then sends them to the requester (Activity 10). Finally, the PM assesses the performance of testers and prepares the final test report for the requester. (Activity 11). In the tester's evaluation, the points and rank are updated based on the quality of the written descriptions for each defect and the number and types of defects found. The final report includes incidents exported, test summaries, and tester feedback and recommendations.

While there are several frameworks and review studies regarding crowdsourcing (e.g., [23–25]), these are not focussed on software engineering [26]. Hence, more research regarding the concepts, advancements, and challenges associated with using crowdsourced software engineering including CST are required [26,27,28].

There is one secondary study about CST [9] in which the author provided a basic bibliometric and descriptive analysis of the literature focusing on revealing reproducible and quantifiable results. That study offered quantification concerning publication time, research methodology, and type of testing used in the literature. The present study differs from that study in that it provides a deeper and more focused qualitative analysis of the research in the field of CST. In particular, it is concerned about the improvements proposed in the literature and the evaluation studies on CST.

Furthermore, one survey [29] and one SLR [30] related to the general domain of crowdsourced software development are found. In addition to the small number of studies about CST (21 papers in [29] and 10 papers in [30]), these studies do not consider how the literature pro-

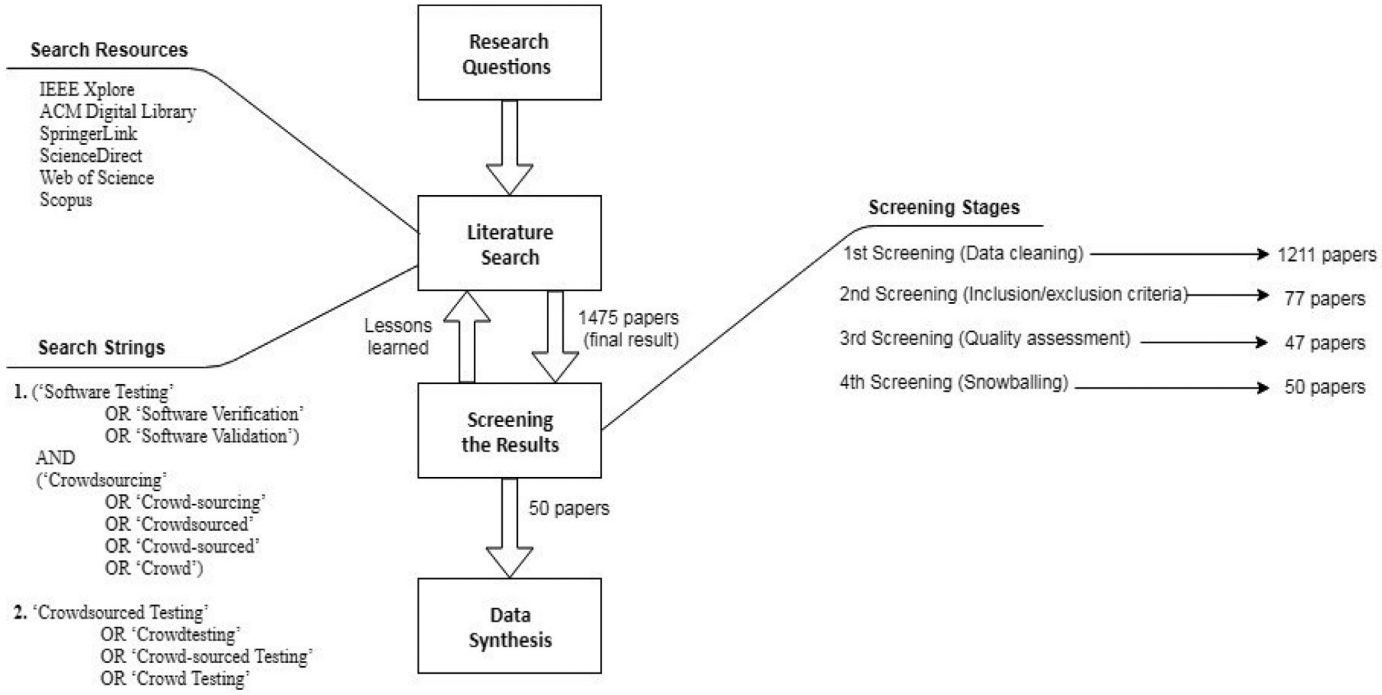


Fig. 2. SLR process.

vides improvement support for the CST process or how it assesses the value of CST application.

3. Research methodology

This research was conducted using the SLR guidelines developed by Kitchenham and Charters [31]. The main steps to create an SLR were followed including defining research questions (RQs), determining search strings and databases, exploring relevant studies, and identifying the final list of key studies. This section describes the details of our research methodology. Fig. 2 also shows the main steps.

3.1. Research questions

CST is growing quickly in practice with dozens of tools and hundreds of thousands of participants [22]. As a result, academia has increasingly been paying attention to this new research area in the last decade, providing a literature covering various aspects of CST. Previous work has focused on providing improvements to specific activities in the CST process and has evaluated the use of CST through many empirical studies.

While previous work presents a concrete value to the body of knowledge, there is still a need to see the big picture in CST research. That is, it would be interesting to know which CST activities have been improved in the recent literature and what is missing, how the solutions provided for a particular CST activity differ from each other, and what challenges and issues are reported through the various studies and under what circumstances. To date, there is no research that has analysed these issues. This motivates our research, which attempts to answer these two RQs:

RQ1: What are the current studies aiming to improve and provide solutions for CST? What kind of process activities does the current research attempt to support?

RQ2: What are the evaluation studies reported for assessing the value of using CST? What are the challenges identified by these studies?

This study represents a checkpoint for analysing the achievements of the research, in terms of how it contributes to the practice of CST by providing improvements and solutions. Doing so can help establish a basis for further research. Additionally, our study sums up the challenges and issues in applying CST, which can help determine the maturity of using CST. This can guide academics to determine what solutions are required and help practitioners understand the obstacles identified through the various studies.

3.2. Literature search

Six popular databases were selected to conduct our SLR: IEEE Xplore, ACM Digital Library, SpringerLink, ScienceDirect, Web of Science, and Scopus. These databases have also been used in other relevant secondary studies [9,29,30] and in SLRs that intersect between crowdsourcing and computing (e.g. [32,33]). These databases are main resources of literature relevant to our study, and thus provide proper confidence of covering relevant publications. Only academic publications were considered in our research and therefore, other types of sources such as blog posts were excluded.

The search was performed without a start date for the papers published until the end of June 2019. A number of strings were used to extract all the published works from the above databases as follows:

- ('Software Testing' OR 'Software Verification' OR 'Software Validation') AND ('Crowdsourcing' OR 'Crowd-sourcing' OR 'Crowdsourced' OR 'Crowd-sourced' OR 'Crowd'),
- 'Crowdsourced Testing' OR 'Crowdtesting' OR 'Crowd-sourced Testing' OR 'Crowd Testing'.

The researcher learned about the relevant key search terms/search strings from other secondary studies [9,29,30]; however, during the snowballing process, we observed that a few publications did not appear in our previous search. Therefore, we used additional search keywords (i.e. 'Crowdtesting' and 'Crowd-Testing') and then searched the databases again.

3.3. Screening the results

The initial search in the six databases produced 1475 results. We screened the results obtained in four main stages (Table 1):

Table 1
Screening stages.

Screening Stage	IEEE	ACM	Springer	Science Direct	Web of Science	Scopus	Total
Extracted	153	154	305	157	24	682	1475
1st Screening	139	116	282	147	11	516	1211
2nd Screening	33	13	5	5	2	19	77
3rd Screening	21	8	4	4	1	9	47
4th Screening (Snowballing)	+ 3						**50**

Table 2
Quality assessment criteria.

Criteria	Question
1	Does the study provide complete primary research (not just abstract,, position paper without results, or review paper)
2	Is the objective of the study clear?
3	Is the methodology/approach reported?
4	Are the study results reported and evaluated?

1st Screening (data cleaning): In this stage, non-scientific, empty, or duplicate results (indexed in multiple databases) are removed. This stage produced 1211 results.

2nd Screening (Inclusion/exclusion criteria): The following are the inclusion criteria adopted in this research. The paper should (i) be relevant to the scope of this research, (ii) be peer-reviewed, (iii) report an original contribution that has not been published elsewhere, and (iv) be written in English. Papers that meet at least one of the following exclusion criteria were eliminated: (i) not relevant, (ii) not peer-reviewed, (iii) different paper titles but reporting similar research results (i.e., in this case, only the most complete one is considered), or (iv) not written in English. This stage resulted in 77 publications.

3rd Screening (Quality assessment): We used knock-out criteria to include only good-quality studies relevant to the RQs. The quality criteria are presented in Table 2. This stage resulted in 47 publications.

4th Screening (Snowballing): We searched the references of the papers identified in the previous stage to find any relevant papers that were not in our list. Three papers were added to the list providing a total of 50 papers (P1, P2, ..P50) ready for investigation (Appendix). Most of the main references used in these papers had already been recorded in the previous stage. This provided us with confidence that the main publications had been listed. It also shows that the area is still relatively young as only 50 papers were found relevant. The extracted results along with the publications included in each screening stage can be accessed through this link.[4]

3.4. Data synthesis

The 50 key studies were synthesised by conducting a thematic analysis. It is a qualitative method of generating patterns or themes from data in a rigorous way [34]. A theme represents a level of abstraction or meaning from the data that is related to an RQ. The classification of the main components of CST [18,20,21] and its process activities [6,22] found in the literature helped build the high-level themes considered in this paper. Apart of this, we used an inductive approach to analyse the data; we read the papers many times, moving back and forward to extract the codes and group common codes to define the themes. The themes for the RQs are presented in the tables and figures in the next section.

The schema was refined continuously while synthesizing the data to consider new data. To double-check our interpretations, we cross-examined how researchers discussed the work of previous primary studies. In addition, we consulted a domain expert, who has publications in the area, to clear ambiguity regarding some data and to test our interpretations developed during synthesis.

The process executed in the thematic analysis is adopted from [34] and [35]. The steps involved in this process with examples of how they were carried out are described below:

- First, all the papers were read at least once to become familiar with the nature of the ideas explored in these papers. It was a free reading without any intention of classifying the papers.
- Second, the papers were read again, but this time we focused on searching for segments of texts that were related to our RQs. The relevant context and results of the papers were highlighted. Samples of some texts are presented in Fig. 3(a).
- Third, the highlighted texts were revisited with the aim of defining a list of codes for each study. Examples of initial coding are illustrated in Fig. 3(b). They include a large numbers of codes associated with each paper and include some redundancies and different levels of granularity. However, this step has been repeated many times. As more papers are investigated, the codes are refined; they are continuously compared with each other and merged when they reflect a similar meaning.
- Fourth, the codes identified in the previous step were grouped into higher categories forming the main themes discovered in the studied papers. To clarify this step, a partial thematic map is given in Fig. 3(c). It shows how low-level labels associated with the selection of suitable testers (activity number 6 in Fig. 1) are grouped into several sub-themes that are themselves grouped into higher-level themes. For example, "test execution time", "recruited number of testers", and "total reward of testers" have all been categorized into "testing cost". With the help of the domain expert, this step included iterative revisions of the allocation of codes to themes and revising the names of themes to make sure they reflect the corresponding codes.

4. Findings

This section presents the findings of this study. In general, the publications covered can be classified as either improvement studies, attempting to advance CST with new improvements and solutions (associated with RQ1), or evaluation studies, reporting evaluations and assessments showing the value of using CST (associated with RQ2). The number of publications for the former (27 papers) is slightly higher than the latter (23 papers), as shown in Fig. 4.

[4] http://github.com/salyahya99/SLR-CST

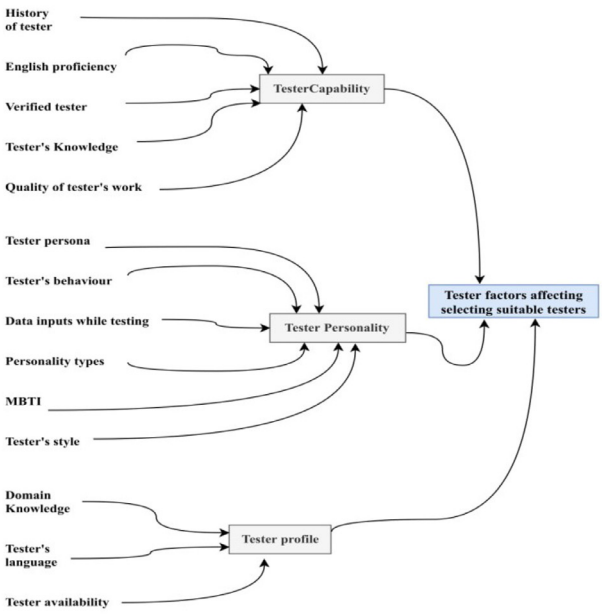
B. Evaluation Framework for Tester's Behaviour

For an effective testing team, it is important that testers are assigned the testing tasks that match their testing style. It is not only important for the task assignment but also for providing appropriate directions to the testers if their testing style do not match the test goals. To study this, first, we manually studied the videos of all the testers to understand various testing behaviors. We propose a tester behavioral evaluation framework that objectively captures testers' behaviors with metrics. A tester's behavior can be explained with respect to two dimensions; first, how he explores the software and second, what kind of data inputs he uses while testing. Hence, a tester persona (aka behavior) is described as a tuple <*Exploration Style, Data Input*> where the first part captures exploration dimension and the second part captures the data input dimension. Based on the exploration dimension, we define following classes of testers:

(a)

	Paper	Codes
2	P15	designs a real-time collaborative testing approach
3		task partition algorithm
4		job assignment problem
5		testers' capability
6		minimizing testing time
7		LOC
8	P16	Worker Recommendation
9		diversity of workers
10		Testing context
11		Domain knowledge
12		Capability
13		machine learning model
14		device model
15		ROM
16		network environment
17		operating system
18	P23	Tester's behaviour
19		allocating testing tasks
20		tester persona
21		exploring software while testing
22		data inputs while testing
23	P26	selecting appropriate testers
24		Personality types
25		Purpose of testing
26		MBTI

(b)



(c)

Fig. 3. Data analysis and synthesis (a) samples of segments of texts, (b) samples of initial codes, (c) partial thematic map.

4.1. RQ1: what are the current studies aiming to improve and provide solutions for CST? What kind of process activities does the current research attempt to support?

The CST process comprises several activities as illustrated in Fig. 1. The results of the investigation on what CST process activities are supported in the current literature by providing improvements and solutions are described in this section. Amongst the process activities of CST, three are supported in the literature (highlighted in Fig. 1). These are: selecting suitable testers (activity number 6), reporting defects (activity number 8), and validating submitted defects (activity number 9). Fig. 5 shows the number of publication for each of these activities.

4.1.1. Selection of suitable testers

A manual method of selecting testers was identified by Deng et al. [P10]. They required testers to have established good records in Android applications development, to have good history of projects finished on time/budget, and to be able to communicate in English. They also verified candidates' identities (i.e. by phone or Facebook) and interviewed

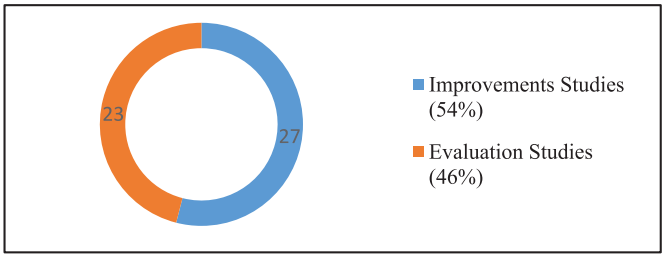


Fig. 4. Number of publications of improvements studies and evaluation studies.

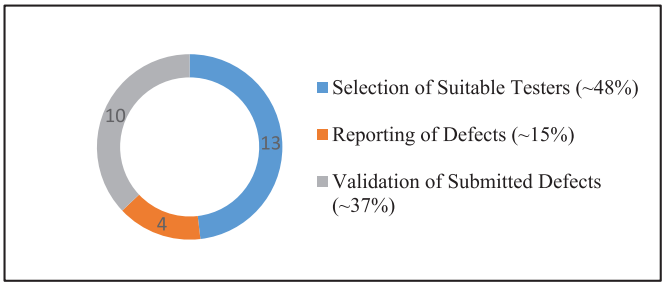


Fig. 5. Number of publications for each of the three supported process activities.

them to determine their knowledge and experience in testing Android applications.

Because the manual process of selecting testers can be costly and time-consuming from a practical point of view especially if the required crowd size is large, a number of automated approaches for selecting suitable testers are proposed. The researchers in [P28] designed a tester selection algorithm, which dispatches a web service test task to one tester out of those with similar running environment, including the same network type. Their work was improved further with Tempo-Spatial Tester Selection in [P29], where the invocation time and location of testers are taken into consideration in the tester selection activity. Xie et al. [P46] contributed to the mobile app testing by providing an approach that considers the variety of operating contexts (such as including heterogeneous devices, various wireless networks, and different locations) when selecting testers.

In [P15], a real-time dynamic assignment algorithm (MMA) to match multi-task testing to the most appropriate testers was proposed. A heuristic matching strategy, where the difficulty level of the test case is compared to the corresponding ability of the tester, was used. In [P16], a machine learning model, which aims to recommend a minimum number of crowd workers who can detect the maximum number of bugs for a crowdsourced testing task, was developed. The matching model was characterised by three dimensions, i.e. testing context, tester capability, and domain knowledge. The testing context contains the device model of a crowd worker, the operating system, read-only memory (ROM) of the device, and network environment. Capability is abstracted from the worker's historical testing information (e.g. the number of reports submitted, and the number and percentage of bugs detected). Domain knowledge represents the domain experience that a crowd worker obtained by performing past testing tasks.

Dubey et al. [P23] observed that testers have different testing styles (i.e. persona), which need to be understood more thoroughly for engaging them better. The authors manually studied videos of testers to understand various testing behaviours. Then, they proposed a tester behavioural evaluation framework that objectively captures the testers' behaviours. A tester's behaviour can be explained in terms of two dimensions: first, how the tester explores the software and second, what type of data inputs he/she uses while testing. In the same way, the purpose of the study in [P26] was to find what tester personality types were suitable for which type of testing techniques used in the crowdsourcing environment. A technique popular for indicating the different types of personality called the Myers-Briggs Type Indicator (MBTI) test [36] was used to classify distinct tester personalities. They concluded that extrovert testers are more suitable for black box testing techniques and testers with introvert personality perform white box testing techniques much better than testers with extrovert personality.

A test case assignment algorithm was designed in [P34] based on a greedy approach for assigning test cases to individual testers. This study modelled collaborative testing in a crowdsourcing environment as a job assignment problem and represented the problem using an integer linear programming formulation. Many factors that could affect testing were considered: test complexity, test case coverage, test execution time, and tester's trustworthiness (which depends on the tester's experience, ability, and availability). Motivated by this work, Guo [P37] proposed a dynamic assignment algorithm for matching each unassigned task to the most appropriate worker at run-time. This can be particularly valuable for large-scale collaborative testing problems.

Cui et al. [P40] proposed a multi-objective crowd worker selection approach that has three objectives: maximising the coverage of test requirement, minimising the cost, and assuring that more experienced workers are selected. It leveraged the evolutionary algorithm, NSGA-II, to optimise the three objectives when selecting workers.

The researchers in [P41] presented a hybrid approach for selecting crowd workers. It comprised three strategies for worker selection: experience, relevance, and diversity strategies. The experience strategy is designed to select experienced workers who have discovered many bugs previously. The relevance strategy involves selecting workers who have expertise relevant to the required test task. The diversity strategy is designed to select workers with diverse expertise.

Leicht et al. [P44] studied the effects of task complexity and user expertise on tester's performance in CST. The results can help understand what types of testing tasks should be assigned to which group of testers (i.e. experts or novices). Consequently, they could decide whether a certain type of testing can be crowdsourced or whether it should rather be carried out using other testing approaches. The task complexity is categorised as simple tasks (surface structure) and complex tasks (deep structure). With simple tasks, testers typically need to focus on user interfaces, as surface structure tasks address defects that can be identified by purely relying on visible inputs and outputs. Complex tasks are concerned with the underlying structures and processes of software where testers need to think about how a software should behave when certain steps or operations are conducted.

As stated in Section 3.4, an inductive approach is considered to discover themes related to the selection of suitable testers. Through this approach, the relevant papers were reviewed line by line and a code is developed as factor affecting the selection emerges. As more papers are investigated, the codes are refined; they are continuously compared with each other and merged when they reflect the same factor. At the end of the review, these factors can be broadly classified into tester and testing factors (Fig. 6). The tester factors include tester capability, tester personality, and tester profile. The testing factors include testing complexity, testing context, testing coverage, and testing cost. To avoid misconceptions of the different terms used in the literature, they have been defined in Table 3; further, in Table 4, we summarized how they are considered by each paper, as discussed above, while the frequency of support in the literature for each factor is presented in Fig. 7.

4.1.2. Reporting of defects

Gómez et al. [P25] used machine-learning techniques to support reproducing mobile app context-sensitive crashes encountered by ordinary users. The approach automatically collects data from crowd devices, identifies recurrent crash patterns caused by user actions and contexts, and generate crash test suites. Furthermore, the authors in [P4] developed a method of making crowdsourced test reports with rich screenshots but insufficient text descriptions that are easier to understand by

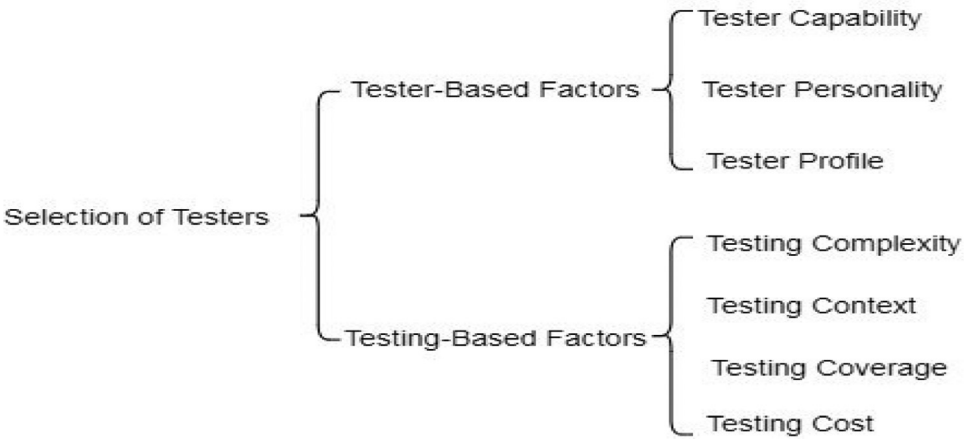


Fig. 6. Classification of factors affecting the selection of suitable testers.

Table 3
Definition of factors affecting the selection of suitable testers.

Factor	Description
Tester Capability	Degree of tester's quality of work (i.e. tester's rating and number of bugs detected)
Tester Personality	The perceived character and behaviour of tester
Tester Profile	Basic information such as tester's language, domain knowledge, and availability
Testing Complexity	Difficulty of a testing task (e.g. number of test cases per web page and number of test cases per line of code)
Testing Context	Test environment such as device, operating system, and network type
Testing Coverage	Selecting test cases for maximal coverage of software (e.g. selecting test cases that cover all web pages of a website)
Testing Cost	The cost in terms of test execution time, required number of testers, etc.

Table 4
Summary of what factors affecting the selection of suitable testers are considered in the literature.

Paper	Selection Mode	Tester-Based Factors			Testing-Based Factors				Additional Information
		Tester Capability	Tester Personality	Tester Profile	Testing Complexity	Testing Context	Testing Coverage	Testing Cost	
[P10]	Manual	√		√					*Tester Capability:* tester must have previous projects finished on time and on budget with a 5-star rating *Tester Profile:* language
[P15]	Automatic	√			√				*Testing Complexity:* lines of code per web page
[P16]	Automatic	√		√		√			*Tester Profile:* time availability *Testing Context:* device model, operating system ROM, and network type
[P23]	Automatic		√						*Tester Personality:* tester style is determined based on how he/she explores the software and what type of data inputs he/she uses while testing
[P26]	Automatic		√						*Tester Personality:* determined by MBTI test
[P28]	Automatic					√			*Testing Context:* network type and location
[P29]	Automatic					√			*Testing Context:* network type and temporal-spatial information of testers
[P34]	Automatic	√		√	√		√	√	*Testing Complexity:* lines of code per web page
[P37]	Automatic	√			√		√	√	*Testing Complexity:* lines of code per web page
[P40]	Automatic	√					√	√	–
[P41]	Automatic	√		√					–
[P44]	Automatic	√			√				*Testing Complexity:* surface structure/complex structure
[P46]	Automatic					√			*Testing Context:* device model, network type, and operating system

generating descriptive keywords for the screenshots of CST mobile test reports. The spatial pyramid matching (SPM) technique [37] was used to measure the similarity of screenshots, and then natural language processing techniques were applied to analyse the text descriptions of the well-written test reports to obtain descriptive keywords. In addition, the research in [P31] improved the quality of inspected test reports by utilising additional useful information contained in duplicate test reports. More specifically, it utilised the text fields of test reports, i.e. environment, input, and description, to combine duplicate test reports that may complement each other from distinct aspects. The authors in [P6] provided a method of summarising a defective report by selecting a set of sentences from its description and comments to conclude the main idea of such report.

The four papers supporting defect reporting are summarised in Table 5. They can be classified into two types: creating test reports from scratch and improving the quality of already written test reports.

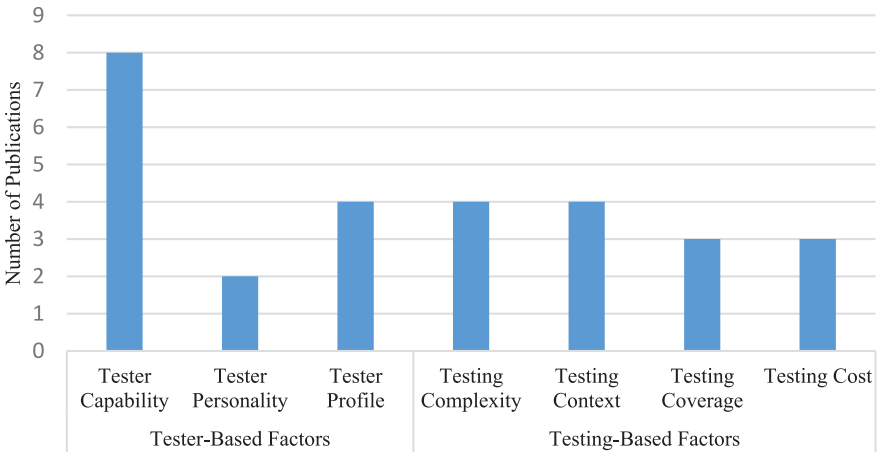


Fig. 7. Frequency of the support in the literature for each factor affecting the selection of suitable testers.

Table 5
Summary of papers supporting defect reporting.

Type	Paper	Aim
Creating test reports from scratch	[P25]	Automatic generation of crash test reports from crowd devices
Improving the quality of already written test reports	[P4]	Enhancing test reports that are rich with screenshots but include little text description by generating descriptive keywords for the screenshots
	[P31]	Enhancing test reports by combining duplicate test reports
	[P6]	Summarising test reports

4.1.3. Validation of submitted defects

The support provided for reporting defects helps produce good test reports. However, with the high number of test reports, solutions are required to accelerate the process of reviewing these reports and validating which of the submitted defects are worth reading.

Chen et al. [P8] attempted to improve test report quality by classifying test reports as either 'good' or 'bad'. They defined four indicators to measure the desirable properties of test reports: morphological indicators, which mainly reveal the surface characteristics of text of a test report (e.g. size, readability, and punctuation) without considering the content; lexical indicators, which try to lexically evaluate the textual content of a test report according to the user-defined term lists (e.g. the number of imprecise terms); analytical indicators, which leverage some specific domain terms to semantically analyse the textual contents of test reports (e.g. the usage of domain terms); and relational indicators, which measure the structured properties of a test report or reveal whether a test report gives each field of information (e.g. itemisation).

Gomide et al. [P49] developed an event-detection algorithm that recognises tester emotions by processing his/her actions from mouse movements or touch events, which can then be used to disqualify non-conforming usability test reports. These reports are correlated with the high mouse movements, which in turn is correlated with hesitation.

Feng et al. [P24] proposed a hybrid approach to prioritise test reports for use in crowdsourced testing. It comprises two prioritisation strategies: risk and diversity strategies. The risk strategy dynamically selects the most risky test report for inspection in each test cycle aiming at prioritising test reports that are most likely to reveal faults. The diversity strategy selects diversified test reports for inspection by maximising the distances to already inspected test reports with the objective of prioritising test reports that are most dissimilar to the already inspected reports to avoid redundant work and reveal different faults. The research considered text similarity only to extract keywords from test reports that are then used to predict failure risks of tests and calculate distances of test reports. The work in [P17] improved this research by considering both text similarity and image similarity. The SPM [37] technique was employed to measure the similarity of screenshots. Similarly, the authors in [P35] proposed a duplicate detection approach, which integrates details from both the screenshots and textual descriptions to detect duplicate crowd testing reports. They extracted four types of features to categorise the screenshots and the textual descriptions (image structure feature and image colour feature, term frequency-inverse document frequency (TF-IDF) feature, and word embedding feature), and then designed an algorithm to discover duplicates based on the four similarity scores obtained from the four features.

Wang et al. [P13] developed an automated test report classification approach in real crowdsourced testing practice. The approach supports classifying true positive reports by solving two problems: local bias and lack of labelled data. Local bias means that the contained technical terms may be shared by only a subset of test reports rather than all the reports. To solve this problem, reports that share more technical terms are used as they are more helpful to build classifiers. In addition, labelling data instances can be difficult, time-consuming, or expensive to obtain and therefore, the authors used active learning to attempt to overcome the labelling bottleneck by selecting unlabelled instances and asking users their labels. This work improved the previous study in [P50], where the authors proposed a cluster-based test report classification approach to effectively categorise CST reports when there are plenty of training data. Because the training data might not always be available, the latter work [P13] can reduce the effort of collecting training data while still achieving good performance. The same authors observed that crowdsourced testing projects belong to a large variety of domains, such as travel, music, and safety, where different technical terms are used in the test reports of different domains to describe the software behaviour. The textual features derived from domains can be significantly different in their distributions. Therefore, in [P14], they proposed a domain-adaptation report classification approach for cross-domain crowdsourced test reports, which can eliminate the data distribution difference across domains.

Jiang et al. [P21] proposed a test report fuzzy clustering framework, which aggregates redundant test reports into a set of clusters to reduce the inspected test reports. The framework contributes to solve particularly these three barriers: invalid barrier (i.e. false positive test reports and empty test reports), uneven barrier (i.e. different descriptive terms for the same bug description), and multi-bug barrier (i.e. testers may report multiple bugs in one test report; thus, they need to be simultaneously partitioned into many clusters).

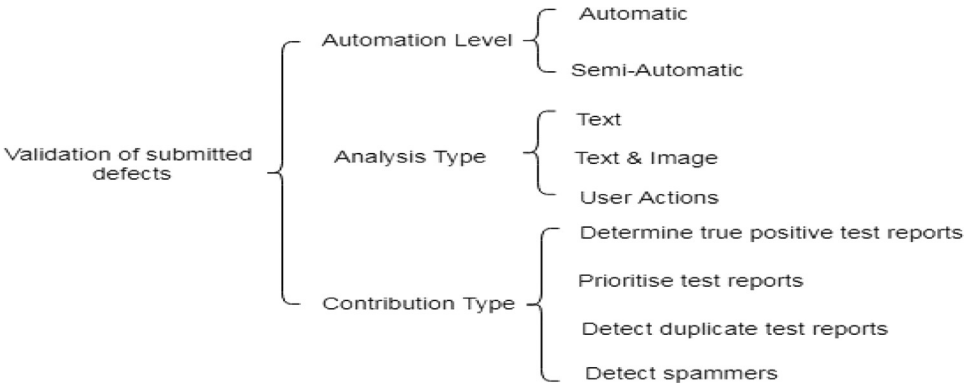


Fig. 8. Classification of the support found in the literature on validation of submitted defects.

Table 6
Classification and summary of the support found in the literature on validation of submitted defects.

Paper	Automation Level	Analysis Type	Contribution Type	Summary
[P8]	Automatic	Text	Determine true positive test reports	Four categories of indicators: morphological (e.g. size, readability, and punctuation), lexical (e.g. the number of imprecise terms), analytical (e.g. the usage of domain terms), and relational (e.g. itemisation)
[P13]	Semi-automatic	Text	Determine true positive test reports	It supports classifying true positive reports by solving two problems: local bias and lack of labelled data
[P14]	Automatic	Text	Determine true positive test reports	A domain-adaptation report classification approach for cross-domain crowdsourced test reports
[P17]	Automatic	Text and image	Prioritise test reports	Text similarity as in [P24] and image similarity (using SPM technique)
[P21]	Automatic	Text	Prioritise test reports	A fuzzy clustering framework to solve invalid barrier, uneven barrier, and multi-bug barrier
[P24]	Automatic	Text	Prioritise test reports	Two prioritisation strategies combined: the risk and diversity strategies
[P35]	Automatic	Text and image	Detect duplicate test reports	Four types of features to categorise the screenshots and the textual descriptions (image structure, image colour feature, TF-IDF, and word embedding)
[P38]	Semi-automatic	Text	Prioritise test reports	An approach for automating the defect classification relying on partially reliable annotators
[P49]	Automatic	User actions	Detect spammers	It recognizes tester emotions by monitoring user's actions, which can then be used to disqualify non-conforming usability test reports
[P50]	Semi-automatic	Text	Determine true positive test reports	Cluster-based test report classification approach to classify test reports when there are plenty of training data

Hernandez-Gonzalez et al. [P38] proposed an approach for automating the defect classification without the supervision of an expert, i.e. utilising multiple partially reliable annotators (novices) only. It uses the concept of "learning from crowds" for defect classification when expert knowledge is missing. The approach is concerned about categorising defects based on their impacts on customers as presented by the orthogonal defect classification taxonomy [38].

The support provided for validating submitted defects can be broadly classified based on the automation level, analysis type, and the contribution type as follows (Fig. 8 and Table 6):

- *Automation level:* all studies except three provided fully automated mechanisms. The remaining three ([P13], [P38], and [P50]) require minimal user involvement.
- *Analysis type:* the studies analysed defect reports with respect to the text, text and image, or user actions during testing.
- *Contribution type:* there are four types of contribution, i.e. determining true positive test reports, prioritising test reports, detecting duplicate test reports, and detecting spammers.

Fig. 9 shows the number of publications for validating submitted defects, classified based on their automation level, analysis type, and contribution type.

4.2. RQ2: what are the evaluation studies reported to assess the value of using CST? What are the challenges identified by these studies?

This section presents the findings related to the evaluation studies of CST. The characteristics of these studies are first described; then the main challenges reported are discussed.

4.2.1. Characteristics of the studies

In total, there are 23 studies presenting an assessment of applying CST (Table 7). All studies except four conducted experiments to evaluate CST. Three of the remaining studies, i.e. [P19], [P30], and [P36], used interviews and questionnaires to evaluate CST and one ([P43]) reported a case study.

The studies are focused on several testing types as indicated in Table 8. It is evident from the results that most of the research is about usability testing. Our interpretation on this fact is that it is the most suitable type that can leverage crowds that often do not have sufficient expertise in software testing and system under test.

The crowd types vary between studies with the most use for crowd from intermediary platforms (e.g. MTurk and CrowdFlower) then students (Table 9). One study ([P27]) used diverse crowd types to investigate the differences between the types. The crowd size for the 19 experimental studies is relatively good: 13 studies had a large crowd size (L) (presuming $L > 40$ persons), 3 studies had a medium size (M) (presuming $10 < M \leq 40$), and 3 studies did not mention the crowd size. Additionally, five studies are solely related to mobile CST, which reflect an increasing interest in using CST for mobile systems. There are various systems under test that are investigated in the studies including e-commerce applications, academic websites, and communication systems.

4.2.2. Identified challenges

Sixteen out of the 23 investigated studies have explicitly mentioned challenges associated with using CST for software testing. The studies that did not explicitly mention any challenges are [P1], [P3], [P7], [P20], [P27], [P33], and [P39].

Table 7
Characteristics of evaluation studies about CST (Legend: NM→ Not mentioned, NA→ Not applicable).

Paper	Main Aim	Evaluation Approach	Testing Type	System Under Test	System Type	Crowd Type	Crowd Size	CST Platform
[P1]	Evaluating a web toolkit that supports automated usability testing with crowdsourcing to enable user testing	Experimental	Usability testing	Wikipedia website	Web	Intermediary crowd	L	New tool integrated with MTurk
[P2]	Assessing the use of crowdsourcing for model verification	Experimental	Model verification	Restaurant application	General	Intermediary crowd	L	CrowdFlower
[P3]	Assessing a gamification-based approach of crowdsourcing to produce better software tests and mutants	Experimental	Unit testing	20 open source Java classes such as sorting integer number class and validating International Bank Account Number class	General	Students	M	New
[P5]	Reporting an evaluation study that compares a new usability tool-based crowdsourced approach to conventional lab testing	Experimental	Usability testing	Academic website	Web	Intermediary crowd	M	New tool integrated with MTurk
[P7]	Presenting a CST service, which supports crowdsourced event traces collection and cross platform execution of tests across the devices of users	Experimental	Compatibility testing Performance testing	Eight mobile apps from AnZhi including social networking and image editing apps	Mobile	NM	NM	New
[P9]	Exploring the possibility of using CST for GUI tests using VMs	Experimental	Usability testing	Multiple, including Xface (editor software) and USB stick mounting	General	Intermediary crowd	L	MTurk
[P11]	Using crowd to verify and fix test cases with synthesised assertions with respect to the documentation	Experimental	Oracle problem	NM	General	Intermediary crowd	NM	MTurk
[P12]	Assessing the cost/benefit of engaging different sizes of mini-crowds to test an intelligent assistant	Experimental	Functional testing	Intelligent assistants	General	Students	L	None
[P18]	Studying if there is an overlap between the types of defects customers find in the field compared to the in-house test teams	Experimental	Functional testing Security testing Performance testing System testing	Collaboration systems	Web	Customer	NM	None
[P19]	Investigating the possibility of integrating crowd and laboratory testing	Interviews/ questionnaires	General	General	Mobile	NA	NA	None
[P20]	Investigating the ability of crowd-based tests to assist the search-based testing tools	Experimental	Test case generation	Google Play Apps	Mobile	Intermediary Crowd	L	MTurk

(*continued on next page*)

Table 7 (*continued*)

Paper	Main Aim	Evaluation Approach	Testing Type	System Under Test	System Type	Crowd Type	Crowd Size	CST Platform
[P22]	Studying the use of CST in educational projects	Experimental	Compatibility testing Usability testing Functional testing	Baidu input (Android App) Baidu browser (web browser) Baidu player (Multimedia player)	General	Students	L	None
[P27]	Exploring the use of CST with different types of crowd	Experimental	Functional testing Usability testing	Industrial production system Insurance system Banking system	General	External test experts Employees Customers	L	NM
[P30]	Studying how crowdsourcing intermediaries manage CST projects and what the associated challenges are	Interviews	General	General	General	NA	NA	testCloud
[P32]	Exploring the effectiveness and challenges of intermediaries in CST	Experimental	Functional testing	5 mobile apps: shopping, social education, music, and management	Mobile	Intermediary crowd	L	Kikbug
[P33]	Studying the impact of crowd size and time constraints on testing	Experimental	Functional testing	Text editor	Desktop app	Students	L	NA
[P36]	Studying the process of CST	Interviews/ questionnaire	General	General	General	NA	NA	None
[P39]	Evaluating how Microsoft leveraged the crowd to conduct performance testing of a large communication system	Experimental	Performance testing	Communication system	General	Employees	L	EI Analytics
[P42]	Evaluating the use of MTurk for usability testing of websites	Experimental	Usability testing	Online store	Web	Intermediary Crowd	M	MTurk
[P43]	Teaching case highlighting the management challenges of an IT department in the financial service inSdustry in designing and introducing an internal CST system	Case study	General	Banking system	General	Employees	NA	None
[P45]	Exploring when firms should use CST as a governance structure	Experimental	Functional testing Usability testing	Mobile banking application Industrial enterprise application	Mobile	Intermediary crowd	L	NM
[P47]	Evaluating crowdsourced usability testing	Experimental	Usability testing	Mozilla Thunderbird Academic website	General	Students	L	None
[P48]	Evaluating the use of crowdsourced usability testing for a graduate school's website	Experimental	Usability testing	Academic website	Web	Intermediary crowd	L	MTurk and CrowdFlower

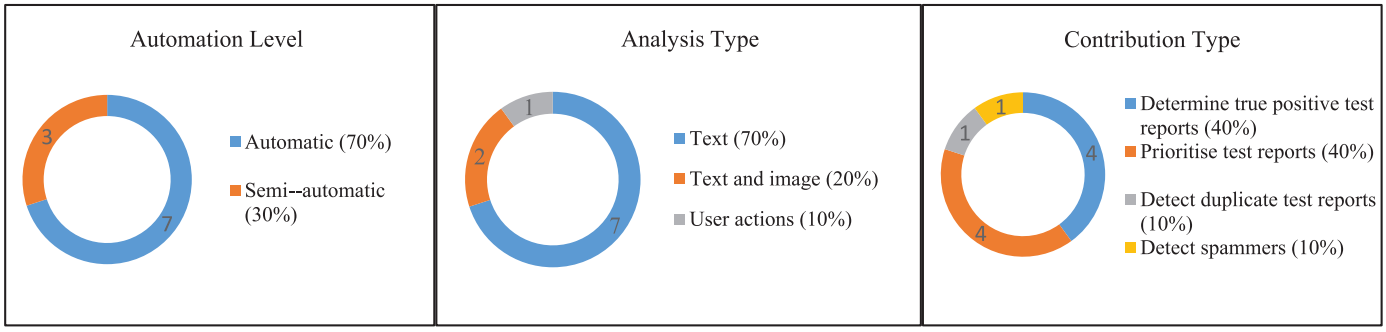


Fig. 9. Frequency of the support in the literature for validation of submitted defects, classified based on their automation level, analysis type, and contribution type.

Table 8
Testing types covered in the evaluation studies.

Testing Activity	No. of Papers	% of Papers	Papers
Usability testing	9	~39%	[P48], [P1], [P5], [P42], [P9], [P22], [P47], [P45], [P27]
Functional testing	7	~30%	[P12], [P18], [P22], [P33], [P45], [P27], [P32]
Performance testing	3	~13%	[P18], [P39], [P47]
Compatibility testing	2	~9%	[P22], [P7]
Model verification	1	~4%	[P2]
Oracle problem	1	~4%	[P11]
Test case generation	1	~4%	[P22]
Security testing	1	~4%	[P18]
System testing	1	~4%	[P18]
General	4	~17%	[P19], [P30], [P36], [P43]

Table 9
Crowd types in the evaluation studies.

Crowd Type	No. of Papers	% of Papers	Papers
Intermediary crowd (i.e., crowd recruited from an intermediary platform)	10	~44%	[P1], [P2], [P5], [P9], [P11], [P2], [P24], [P32], [P45], [P48]
Students	5	~22%	[P3], [P12], [P22], [P33], [P47]
Employees	3	~13%	[P27], [P39], [P43]
Customers	1	~4%	[P27]
External testers	1	~4%	[P27]

Shinsel et al. [P12] investigated the value of engaging a small crowd of end users for the testing of an intelligent assistant from a cost-benefit perspective. The results demonstrated that a mini-crowd of testers provides many advantages besides the decrease in workload. However, these advantages did not scale linearly as the mini-crowd size increases; there is a point of diminishing returns.

Because small software development teams normally have a limited test resource, leveraging crowdsourced customers can help in defect detection. The authors in [P18] investigated if there is an overlap between the types of defects customers find in the field compared to in-house test teams. They found that performance and security defects are specialist focus areas, which are best handled by in-house teams. Further, customers are skilled at finding normal functional defects.

Chen and Luo [P22] explored the value of recruiting a crowd of students in a class to test three commercial platforms. They found that financial rewards are the least motivation to participate in CST projects. The authors pointed out that it would not be easy to attract students to participate in CST after they finish their study modules. This indicates that people may have different motivations to participate in CST; thus, it would be challenging to understand what types of incentives are most appropriate.

Bruun and Stage [P47] compared a CST approach where testers received minimal training to a barefoot approach where software development practitioners received intensive training for conducting usability testing. The results showed that the CST approach suffers from a significantly low detection rate. The authors attributed this to the limited understanding of what usability testing is all about. The same comment on low detection rate was also reported by Winkler et al. [P2].

The challenge of achieving good quality of test results has been raised by many studies. In this challenge, the problem is not only associated with the small number of discovered defects but also on the frequency of false positive defects identified by crowdsourced testers. Liu et al. [P48] evaluated the potential of using crowdsourced usability testing for a graduate school's website using a crowdsourcing platform. They compared the results with a similar classical lab usability test on the same website and found that the quality of test reports produced by the lab usability testing is better than that coming from crowdsourced testers. Similarly, Winkler et al. [P2] presented an experiment about the use of CST for model verification. The results showed that 56% of reported defects are false positive. While the authors did not investigate whether the test reports include spams, Liu et al. [P48] showed that a considerable number of participants recruited from MTurk and CrowdFlower were spammers (18 out of 105 participants). They also highlighted the need for extra effort to write test tasks that are understandable to the crowd. Likewise, Pastore et al. [P11], who focused on using the crowd to verify and fix test cases with synthesised assertions with respect to the documentation, reported that test cases need to be easily understood to keep the human oracle cost to a minimum. They also found that the workers can perform well at verifying CrowdOracles, but their performance depends on the qualification of the crowd and the clarity and complexity of the tests and documentation.

In [P5] and [P48], the authors pointed out that testers do not give many additional comments (in written form) compared to lab testers. This can be a negative aspect, as the message may not be delivered well. In relation to this, two primary studies ([P42] and [P48]) mentioned that more detailed recommendations are not provided by the crowd. They

reported that, different from lab testing, multiple rounds of usability testing are required to complete testing tasks.

Furthermore, some types of testing may require testers to have good Internet connection speed. Dolstra et al. [P9] found that many workers who participated in GUI testing through instantiating virtual machines to run the system under test had long completion times of test tasks. The reason is that many of the testers were from India with poor connection speed.

Guaiani and Muccini [P19] conducted a number of interviews as well as a questionnaire with a crowd to investigate the possibility of integrating crowd-based testing with conventional lab-based testing. The individuals in the crowd were directly asked about the main challenges they face when doing CST. They mentioned that time pressure and competition pressure are amongst the top challenges. From our point of view, this is likely happening due to the nature of the business models of intermediaries that offer test tasks as contests. Similar to [P11], the crowd in this study also mentioned that the complexity of tests is one of the challenges. In particular, they identified these test types as problematic to them: unit/integration testing, security testing, and stress testing. More interestingly, half of the testers participating in the questionnaire mentioned that the amount and type of information they receive as input to run their testing tasks are insufficient. This conforms to the results found in [P48] and [P11].

Leicht et al. [P45] explored when firms should use CST in two case studies: one with a mobile banking application and the other with an industrial enterprise application. They concluded that CST might not be always advantageous. This is the case especially if the testers require functional (not testing) expertise knowledge to test the application or to conduct specific testing types (e.g. security testing). In addition, testing the bank application revealed that there are many data security concerns. The higher the security standards, the more difficult crowdsourcing becomes. Moreover, the authors observe that crowdsourcing can be costly for small testing departments and unsuitable for systems with low maturity or those that require hardware-specific testing.

Gao et al. [P32] explored the effectiveness and challenges of crowdsourced mobile application testing by conducting a large study involving crowdsourcing of five Android applications to 258 crowd testers, who reported 1013 defects in these applications through the Kikbug CST platform. Although a large number of defects were reported, the authors found that many of them are incorrect or meaningless due to what they believe as the lack of domain knowledge. In addition, there is a large number of duplicated bugs. Furthermore, the feedback they received from the crowd showed that 40.3% of them believed that the incentive scheme was extremely low and that the rewards were fixed regardless of the efforts required to find defects. The authors in this study believed that this may cause the crowd to consider that the number of defects is the most important factor in computing the rewards. Therefore, to increase the possibility of winning more rewards, the crowd concentrate on the easy-to-detect defects as opposed to spending time on a few hard-to-detect defects.

Alyahya and Alrugebh [P36] reviewed the process used in eight CST platforms starting from receiving requester's requirements and ending with evaluating testing reports. Understanding the current process is then utilised to identify a set of possible challenges in the process. First, crowd managers (they normally have a dual responsibility: one is related to discussing and defining the requirements with the requester and the other is related to delivering testing tasks to the crowd and obtaining the results from them) are assigned manually in CST platforms. While it is common that one crowd manager supervises multiple projects, the manual process used to allocate crowd managers can be inefficient (i.e. some crowd managers are overworked while others are waiting for new assignments). Second, the testing team is formed manually. Revising the profiles of interested crowd testers and matching their capabilities with the test requirements are indeed time-consuming especially for large projects that can have dozens of candidate testers. Third, there is no available method for monitoring the progress of crowd testers. Some of them may accept to join a testing task but then disappear or do not submit any test report.

Zogaj et al. [P30] carried out three in-depth interviews with the three founders of testCloud, a CST platform. The aim of their study was to achieve better understanding of how CST intermediaries manage the process in a crowdsourcing model. The findings showed that testCloud faces three main challenges: managing the process, managing the crowd, and managing the technology. Managing the process includes finding appropriate mechanisms to define testing requirements with customers, the risk of receiving valueless outcomes from the crowd, and the involvement of customers in the process, for instance, enabling them to monitor the testing progression, to alter requirements as desired, and to countercheck submissions. The challenge of managing the crowd involves the manual allocation of testing assignments. Finally, managing the technology includes facilitating the received testing requirements from customers through the CST platform.

Knop and Blohm [P43] described a case study highlighting the management challenges of an IT department in the financial service industry that was designing an internal crowd testing system. After months of running the new work model, these challenges were reported: supervisors reluctant of employee's participation, inadequate IT competence of employees, parallelism of work modes (the new testing responsibilities contradicts with the employees' conventional everyday work), incentives are inappropriate, and defined testing requirements are unsuitable for the individuals.

Table 10 summarises the 27 challenges identified by the investigated studies. Considering the process of CST illustrated in Fig. 1, the challenges are associated with all the three actors represented in the process: requester (challenges 1–6), crowd (challenges 7–14), and intermediary (challenges 19–27) as well as the input to the process, testing requirements (challenges 15–18). Mapping the challenges with the process activities would not be straightforward because each challenge may be potentially associated with several activities. For example, the 'low defect detection rate' challenge is a requester's challenge as they evidently wish to receive several defects when they perform the activity 'confirm reviewed defects'; however, the source of the challenge could be associated with other activities such as selecting suitable testers. Moreover, although the challenges are distinct, it is clear that they involve many interdependencies (i.e., some of them can cause the emergence of others). For example, the lack of appropriate mechanisms to control spammers (challenge 27) can cause low quality of testing results, i.e., high false positive defect rate (challenge 3). Moreover, the time pressure (challenge 9) affects the defect detection rate (challenge 2).

Amongst the identified challenges, challenge 15 (Complexity of testing tasks) and challenge 7 (Clarity of testing tasks) followed by challenge 2 (Low defect detection rate), challenge 3 (Low quality of testing results - high false positive defect rate), and challenge 13 (Incentive scheme is low or inappropriate) are the challenges frequently mentioned in the literature.

5. Discussion

5.1. Interpretations of the findings

RQ1: *What are the current studies aiming to improve and provide solutions for CST? What kind of process activities does the current research attempt to support?*

Based on the findings related to RQ1 (Section 4.1), it is found that there are three main process activities that the current studies aim to improve: selection of suitable testers, reporting of defects, and validation of submitted defects. These are the process activities that the literature has developed improvements for amongst the ones involved in the CST process (Fig. 1). Other process activities including submitting a testing project by a requester, assigning the project to project managers, eval-

Table 10
Summary of challenges identified by the evaluation studies.

Type	No.	Challenge	No. of Papers	Paper
Requester	1	Diminishing returns (the benefit of CST does not scale linearly with the cost of increasing the size of crowd)	1	[P12]
	2	Low defect detection rate	3	[P2], [P32], [P47]
	3	Low quality of testing results (high false positive defect rate)	3	[P2], [P30], [P48]
	4	Lack of crowd's experience in software testing	2	[P11], [P43]
	5	Clarity of testing reports	2	[P5], [P48]
	6	The need for multiple rounds of testing	2	[P42], [P48]
Crowd	7	Clarity of testing tasks	4	[P11], [P19], [P43], [P48]
	8	Infrastructure challenges (e.g. speed of Internet connection)	1	[P9]
	9	Time pressure	1	[P19]
	10	Competition pressure	1	[P19]
	11	Lack of knowledge about the system under test	2	[P32], [P45]
	12	Lack of knowledge/experience about software testing	2	[P11], [P43]
	13	Incentive scheme is low or inappropriate	3	[P22], [P32], [P43]
	14	Rewards do not consider that effort vary depending on the defects	1	[P32]
Testing	15	Complexity of testing tasks	4	[P11], [P18], [P19], [P45]
	16	Data security concerns related to the system under test	1	[P45]
	17	Maturity of system under test	1	[P45]
	18	The need of hardware-specific testing	1	[P45]
Intermediary	19	Lack of proper mechanisms to avoid duplicated test reports	1	[P32]
	20	Manual assignment of crowd manager	1	[P36]
	21	Manual process of selecting testers	2	[P30], [P36]
	22	Lack of mechanisms to monitor testers during test cycle	1	[P36]
	23	Lack of appropriate mechanisms to define testing requirements with requesters (e.g. dividing a test job into small test tasks)	1	[P30]
	24	Involvement of requester (i.e. customer) in the testing process	1	[P30]
	25	Manual process of receiving testing requirements from requesters (i.e. customers)	1	[P30]
	26	Supervisors reluctant of employee's participation in the case of internal CST	1	[P43]
	27	Lack of mechanisms to control spammers	1	[P48]

uating the performance of the tester, and preparing final test reports to requesters have thus far not been explored in the literature.

The literature has discussed why selecting suitable testers is applicable with significant impact in the context of CST. In fact, although identifying appropriate workers (worker recommendation) for a particular software development task has extensive been discussed in the literature, new requirements in CST are needed, such as not every crowd worker should be involved in a crowdsourcing testing task because of the uncertainty, largeness, and undefinedness of the crowd [39] [P15] [P16]. In addition, the recommended testers can be dependent on each other's work; thus, testers' performance together may influence the final test output [P16]. Therefore, more effort is required to identify the most suitable testers. Selecting inappropriate testers not only reduces the quality of test output but also raises the cost of hiring them [P40].

As Xie et al. [P46] observed, selecting testers is still challenging in most CST platforms. Therefore, the current research studies provide valuable support in this regard. The current studies considered several crucial factors affecting the selection process from two angles, namely tester-based factors (tester capability, tester personality, and tester profile) and testing-based factors (testing complexity, testing context, testing coverage, and testing cost), both giving a total of seven different types of factors.

Amongst all factors, tester capability is most considered in the literature (8 papers). This is not surprising since the uncertainty of a crowd forms a fundamental challenge in crowdsourcing paradigm that could be mitigated by considering worker capability [39]. In contrast, tester personality factor is least considered (2 papers), perhaps due to the cost associated with discovering tester personality in addition to the limited applicability (i.e., it could be useful only in few types of testing). Indeed, while a few of the factors such as tester capability and tester profile may affect the selection of testers in almost all testing jobs, others depend heavily on the nature of the system under test and the type of testing required. For instance, it is essential to consider the testing context factor for a testing job that is required for specific hardware or software. The findings suggest that industry practitioners such as CST project managers should think of more factors that may help select the most appropriate testers for a testing job.

For defect reporting, the literature highlighted the impact of providing improvements that support this activity in the context of CST. Chen et al. [P31] stated that CST recruits not only professional software testers, but also ordinary end users who are inexperienced in software testing. An investigation on real industrial data shows that test reports are produced in descriptive natural language. These reports lack important details due to the poor performance of workers and the inconvenience of editing over mobile devices (i.e., which is common for mobile testing); thus, test requesters may find it difficult to understand the defects. Furthermore, Liu et al. [P6] mentioned that because CST normally involves several workers, the number of submitted test reports can be overwhelming. Reviewing and understanding these test reports would be extensively time-consuming.

There are only four primary studies aiming to improve the defect reporting activity. This significantly small number implies that the contribution to this activity is limited and more research is required. A possible gap to improve is to think of how to build defect reporting systems that best support ordinary testers. There are many workers who are not specialised in software testing or have little experience in this domain but can help evaluate some testing activities such as usability and functional testing. Such workers likely require support through building simple but intelligent reporting systems, which can help them predict the ideas they would like to deliver, measure the quality of test reports before submission, provide suggestions to improve the reports, and formalise the reports in a way that expedites reading and evaluating these reports by developers/CST project managers. We believe that incorporating recommendation and readability techniques [40–43] in defect report systems can play a key role in improving the quality of the delivered test reports. Moreover, there is only one study [P25] that supports automatic generation of reports from crowd devices. The approach of generating crash test reports produced in [P25] can be applied to other types of testing,

for instance, for GUI monitoring through generating monitored usage logs to be replayed when a defect occurs [44].

Similar to selecting suitable testers and defect reporting, it is also important to support validating test reports in CST. A characteristic of CST is that it is strictly limited in time (i.e., few days) [12] and that several test reports are regularly submitted, and the available resources to review them are insufficient [P8] [P13]. Therefore, there is an explicit need to reduce the cost and time required for validating these reports. In addition, because of the uncertainty of workers and variability of their experience in software testing, it is important to provide solutions built particularly for CST such as detecting spammers and determining true positive test reports.

The literature supports improving CST through developing techniques for validating test reports received from the crowd. There is a variety of contribution types, including determining true positive test reports, detecting spammers, prioritising test reports, and detecting duplicate test reports. This indicates more interest and better maturity of the research to improve this activity, as compared with the research on defect reporting activity that is apparently limited. However, these contributions focus mainly on the content of the test report and ignore testers. There is no research that takes into consideration the rating/reputation of testers when filtering the test reports. We believe that it is worthwhile to consider the performance history of testers along with the content of test results when validating the reports. Considering the rating/reputation of testers is particularly useful in intermediary platforms that have large numbers of projects where the same testers frequently submit test reports across projects. Other well-known approaches for quality control in crowdsourcing can also be used in CST such as ground truth, output agreement, and majority consensus [45,46]. Furthermore, there is only one study that considers detecting spammers in CST. It utilises user's actions such as mouse movements and touch events. Because this is a general problem in crowdsourcing, CST can benefit from other spam detection techniques identified in the crowdsourcing literature (e.g. [47,48]).

RQ2: *What are the evaluation studies reported for assessing the value of using CST? What are the challenges identified by these studies?*

Based on the findings related to RQ2 (Section 4.2), it is found that there are 23 CST evaluation studies. Most of them have large and real crowds, which denote the quality of the studies. These studies have identified 27 different challenges, and the author classified them to four dimensions: requester, crowd, testing, and intermediary.

We argue that challenge 2 (Low defect detection rate) and challenge 3 (Low quality of testing results) are the core challenges not only amongst the requesters' challenges but the whole list of challenges. This is because those two challenges are regarding the final outcome submitted to requesters: defects. We believe that the contribution toward mitigating other challenges could ultimately help mitigating these two core challenges.

Another point in the requester's challenges is that if time matters to requesters, they should be aware that the reputation of CST in conducting rapid testing is not applicable with all types of testing. A few types such as usability testing could be lengthy as they may require multiple rounds of testing (challenge 6). This has been reported in two studies that used CST for this type of testing (i.e., [p42] and [P48]).

Regarding the crowds' dimension, challenge 7 (Clarity of testing tasks) and challenge 12 (Lack of knowledge/experience about software testing) have a strong impact on testing. Studies have shown that these considerably affect the performance of the crowd, which by itself affects the quality of testing [P11] [P43]. Although other challenges are still relevant, they are either likely to be less frequent (such as challenges 8 and 11) or can be addressed through managerial decisions (challenges 9, 10, 13 and 14).

Similarly, regarding testing dimension, challenge 15 (Complexity of testing tasks) is the most frequently identified in the literature. Project managers invest a lot of effort to match testing tasks with crowd expertise owing to this challenge. The other challenges in this dimension (challenge 16, 17, and 18) could in fact restrict the use of CST since they deal with specific circumstances related to the systems being tested (i.e., systems with data security issue, less mature systems, and hardware-dependant systems).

Further, the list of challenges associated with the intermediary process shows that the manual approach to deal with the large crowd and high number of test reports is a big obstacle. With the increasing demand on CST services in the next few years, it is expected that the intermediary companies are likely to find it difficult if they continue using ad-hoc solutions to manage their own processes.

Recognising the challenges is crucial to industry practitioners and academic researchers. For industry practitioners, it would be important to be aware of these challenges, assess if they influence their individual situations, determine the impact of these challenges, and develop mitigating measures. Some challenges such as the time pressure affecting crowd performance can be mitigated by simple decisions. However, some others such as the low defect detection rate can involve several correlating factors conferring huge complexity.

For academic researchers, many of the identified challenges represent research gaps that they can support, although few of these challenges already have some solutions for mitigation as illustrated in Section 4.1 (namely challenges 3, 19, 21, and 27). In fact, comparing the three improvement areas (selection of suitable testers, reporting of defect, and validation of submitted defects) with the list of challenges shows that the research community focuses on a tiny angle of the big picture. In particular, computer science researchers can think of the uninvestigated challenges on the intermediary dimension as they can be mitigated by automating many aspects involved in the CST process such as assignment of crowd managers, monitoring testers during test cycles, and receiving testing requirements from requesters.

We believe that many of these challenges require deep understanding and analysis especially that because of the infancy of the research domain, most of them are mentioned few times in the literature. Therefore, further studies about the challenges are required. Additionally, it would be worth investigating how some of these challenges could affect others. For example, how could the lack of mechanisms to monitor testers during the test cycle (challenge 22) affect the low defect detection rate (challenge 2)? What if we add some mechanisms to monitor and support testers before they submit their reports; would this improve the defect detection rate? Is the cost of monitoring testers worth the improvement in defect detection rate?

5.2. Threats to validity

Our findings in this SLR may have been affected by several threats. Kitchenham and Charters [31] mentioned that any SLR should include a discussion about the threats to validity. We categorised the threats of this research as recommended by Zhou et al. [49] based of internal, construct, external, and conclusion.

Internal validity: This type covers how rigorous the SLR process is implemented (e.g. the used search terms, quality assessment, search method, and data synthesis). In this research, the guidelines to conduct a rigorous process identified by Kitchenham and Charters [31] were followed. The details regarding selecting the search terms, screening the results including defining inclusion/exclusion criteria and quality assessment, and the final number of primary studies were reported. With regard to the search terms, we learned about some search terms appropriate for this research from other secondary studies [9,29,30] as well as the snowballing process as clarified in Section 3.2.

Construct validity: This validity is concerned about the selection of libraries and databases, screening criteria, and RQs. The RQs were refined twice during the study to make sure that they reflect exactly what this research aims to. The judgement on the quality of papers was

performed by only one person. This would be a threat because the likelihood of misjudgement would be higher compared to that when more persons are involved. However, the author strove to read the necessary parts of all the candidate papers in the second and third screening process multiple times before making a decision on their quality. This is an exhaustive effort that by itself could cause mistakes; therefore, there was a time gap between the cycles of revision. Moreover, the quality assessment questions tend to be mostly clear and straightforward such as if the approach is reported or not and if the results are reported or not. Finally, in this regard, the author participated in publishing five papers in the area of using crowdsourcing in software development; three of them are particularly related to CST. Therefore, the author has extensive knowledge about many publications in the area before starting this research.

External validity: This is related to the generalisability of the findings obtained from the studies and the accessibility of studies and databases. The SLR utilized six popular databases in computer science, which are commonly used in other SLRs in this discipline. The use of multiple databases reduces the likelihood of subjective errors that could occur during the search. However, more effort was required to filter the papers as many papers were duplicated within the databases. The database names are presented in Section 3.2 and the studies are listed in the Appendix.

Conclusion validity: This type of validity is about the conclusions and interpretations of the findings. In this research, a thematic analysis guidelines introduced by [34] and [35] were used to do an inductive coding. We believe that the proposed codes form a comprehensive representation of the findings of the studies with respect to the RQs. For instance, the study has classified how the literature considers improving the process of selecting suitable testers to tester factors and testing factors. This sort of classification can help practitioners and researchers understand the underlying variables that make contributions differ from each other. However, there is no definite guarantee that the findings on the selected 50 primary studies are generalised to the whole literature because this SLR complies with a set of quality criteria that have resulted in rejecting some studies. In addition, this SLR does not aim at providing statistical and descriptive analysis of the studies but rather focuses on answering specific RQs associated with defining and understanding themes and constructs that are included in and across the studies.

6. Conclusions

This study presented an SLR on the use of crowdsourcing in software testing. It focused on investigating two aspects: first, what improvements and solutions are proposed in the area of CST, and second, what evaluation studies are published and what challenges are reported by these studies. To obtain answers to these questions, 50 papers were rigorously selected and reviewed. It was found that the current literature has 27 studies concentrating on providing solutions to three main process activities: selection of suitable testers (13 studies), reporting of defects (4 studies), and validation of submitted defects (10 studies). The findings also showed that there are 23 evaluation studies that identified 27 different challenges associated with CST. These challenges are categorised into four dimensions: requester, crowd, testing, and process.

This study helps understand the maturity of the CST field. The findings show that the field is fairly young (i.e., earliest paper [P12] is published in 2011) but has rapidly gained increasing interest, yet more research is required as stated in the discussion section.

This study helps the interested readers recognise what has been completed by the CST research community thus far; further, by describing the current progress of CST, it helps understand how future research in this area can be conducted. Knowing the improvements achieved for specific process activities in CST in the current research enables the identification of other process activities that need to be explored. Similarly, knowing the characteristics of the evaluation studies can direct us on what other evaluation studies are worth investigating. Additionally, many of the challenges identified by these evaluation studies represent research problems that need more understanding and require solutions for mitigation.

Furthermore, this research offers opportunities for practitioners, especially those interested in developing advanced CST platforms, to understand what new solutions have been proposed in the literature and the variations between them. It also provides awareness to related parties (i.e. test requesters, testers, and managers of intermediary platforms) on the challenges reported in the literature, which they may encounter during CST tasks.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The author would like to extend his sincere appreciation to the Deanship of Scientific Research at King Saud University for its funding this Research - Group No (RGP-1436-039).

Appendix. Research Studies included in the SLR

[P1] M. Nebeling, M. Speicher, and M. C. Norrie, “CrowdStudy: General toolkit for crowdsourced evaluation of web interfaces,” *EICS 2013 - Proc. ACM SIGCHI Symp. Eng. Interact. Comput. Syst*, pp. 255–264, 2013.

[P2] D. Winkler, M. Sabou, S. Petrovic, G. Carneiro, M. Kalinowski, and S. Biffl, “Improving Model Inspection with Crowdsourcing,” *in Proceedings of the 4th International Workshop on CrowdSourcing in Software Engineering*, 2017, pp. 30–34.

[P3] J. M. Rojas, T. D. White, B. S. Clegg, and G. Fraser, “Code Defenders: Crowdsourcing Effective Tests and Subtle Mutants with a Mutation Testing Game,” *in Proceedings of the 39th International Conference on Software Engineering*, 2017, pp. 677–688.

[P4] D. Liu, X. Zhang, Y. Feng, and J. A. Jones, “Generating descriptions for screenshots to assist crowdsourced testing,” *in 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2018* - pp. 492–496.

[P5] F. Meier, A. Bazo, M. Burghardt, and C. Wolff, “Evaluating a web-based tool for crowdsourced navigation stress tests,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8015 LNCS, no. 4, pp. 248–256, 2013.

[P6] H. Jiang, X. Li, Z. Ren, J. Xuan, and Z. Jin, “Toward Better Summarizing Bug Reports With Crowdsourcing Elicited Attributes,” *IEEE Trans. Reliab.*, vol. 68, no. 1, pp. 2–22, 2019.

[P7] G. Wu, Y. Cao, W. Chen, J. Wei, H. Zhong, and T. Huang, “AppCheck: A Crowdsourced Testing Service for Android Applications,” *in Proceedings - IEEE 24th International Conference on Web Services, ICWS*, 2017.

[P8] X. Chen, H. Jiang, X. Li, T. He, and Z. Chen, “Automated quality assessment for crowdsourced test reports of mobile applications,” *in 25th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER - Proceedings*, 2018, pp. 368–379.

[P9] E. Dolstra, R. Vliegendhart, and J. Pouwelse, “Crowdsourcing GUI tests,” *in Proceedings of the IEEE 6th International Conference on Software Testing, Verification and Validation*, ICST, 2013.

[P10] L. Deng, J. Offutt, and D. Samudio, “Is mutation analysis effective at testing android apps?,” *in Proceedings of the IEEE International Conference on Software Quality, Reliability and Security, QRS*, 2017, pp. 86–93.

[P11] F. Pastore, L. Mariani, and G. Fraser, “CrowdOracles: Can the crowd solve the oracle problem?,” *in Proceedings of the IEEE 6th International Conference on Software Testing, Verification and Validation, ICST*, 2013, pp. 342–351.

[P12] A. Shinsel et al., "Mini-crowdsourcing end-user assessment of intelligent assistants: A cost-benefit study," *in Proceedings of the IEEE Symposium on Visual Languages and Human Centric Computing, VL/HCC*, 2011, pp. 47–54.

[P13] J. Wang, S. Wang, Q. Cui, and Q. Wang, "Local-based Active Classification of Test Report to Assist Crowdsourced Testing," *in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 190–201.

[P14] 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*, 2017, pp. 83–92.

[P15] S. Guo, R. Chen, and H. Li, "A Real-Time Collaborative Testing Approach for Web Application: Via Multi-tasks Matching," *in Proceedings of the IEEE International Conference on Software Quality, Reliability and Security-Companion, QRS-C*, 2016, pp. 61–68.

[P16] J. Wang et al., "Characterizing Crowds to Better Optimize Worker Recommendation in Crowdsourced Testing," *IEEE Trans. Softw. Eng.*, 2019.

[P17] Y. Feng, J. A. Jones, Z. Chen, and C. Fang, "Multi-objective Test Report Prioritization Using Image Understanding," *in Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*, 2016, pp. 202–213.

[P18] D. Malone and J. Dunne, "Social dogfood: A framework to minimise clouc field defects through crowd sourced testing," *in 28th Irish Signals and Systems Conference, ISSC*, 2017.

[P19] F. Guaiani and H. Muccini, "Crowd and Laboratory Testing Can They Co-exist?: An Exploratory Study," *in Proceedings of the Second International Workshop on CrowdSourcing in Software Engineering*, 2015, pp. 32–37.

[P20] K. Mao, M. Harman, and Y. Jia, "Crowd Intelligence Enhances Automated Mobile Testing," *in Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017, pp. 16–26.

[P21] H. Jiang, X. Chen, T. He, Z. Chen, and X. Li, "Fuzzy Clustering of Crowdsourced Test Reports for Apps," *ACM Trans. Internet Technol.*, vol. 18, no. 2, pp. 18:1–18:28, 2018.

[P22] Z. Chen and B. Luo, "Quasi-crowdsourcing Testing for Educational Projects," *in Companion Proceedings of the 36th International Conference on Software Engineering*, 2014, pp. 272–275.

[P23] A. Dubey, K. Singi, and V. Kaulgud, "Personas and Redundancies in Crowdsourced Testing," *in Proceedings of the 12th International Conference on Global Software Engineering*, 2017, pp. 76–80.

[P24] Y. Feng, Z. Chen, J. A. Jones, C. Fang, and B. Xu, "Test report prioritization to assist crowdsourced testing", *in Proceedings of the 10th Joint Meeting on Foundations of Software Engineering* 2015.

[P25] M. Gómez, R. Rouvoy, B. Adams, and L. Seinturier, "Reproducing context-sensitive crashes of mobile apps using crowdsourced monitoring", *IEEE/ACM International Conference on Mobile Software Engineering and Systems (MOBILESoft),* 2016.

[P26] Z. U. Kamangar, U. A. Kamangar, Q. Ali, I. Farah, S. Nizamani, and T. H. Ali, "To Enhance Effectiveness of Crowdsource Software Testing by Applying Personality Types," *in Proceedings of the 8th International Conference on Software and Information Engineering*, 2019, pp. 15–19.

[P27] N. Leicht, I. Blohm, and J. M. Leimeister, "Leveraging the Power of the Crowd for Software Testing," *IEEE Softw.*, vol. 34, no. 2, pp. 62–69, 2017.

[P28] M. Yan, H. Sun, and X. Liu, "iTest: Testing Software with Mobile Crowdsourcing," *in Proceedings of the 1st International Workshop on Crowd-based Software Development Methods and Technologies*, 2014, pp. 19–24.

[P29] M. Yan, H. Sun, and X. Liu, "Efficient Testing of Web Services with Mobile Crowdsourcing," *in Proceedings of the 7th Asia-Pacific Symposium on Internetware*, 2015, pp. 157–165.

[P30] S. Zogaj, U. Bretschneider, and J. M. Leimeister, "Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary," *J. Bus. Econ.*, vol. 84, no. 3, pp. 375–405, 2014.

[P31] X. Chen, H. Jiang, Z. Chen, T. He, and L. Nie, "Automatic test report augmentation to assist crowdsourced testing", *Front. Comput. Sci.*, vol. 13, no. 5, pp. 943–959, 2019.

[P32] R. Gao, Y. Wang, Y. Feng, Z. Chen, and W. Eric Wong, "Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing," *Empir. Softw. Eng.*, vol. 24, no. 2, pp. 537–561, 2019.

[P33] M. V Mäntylä and J. Itkonen, "More testers – The effect of crowd size and time restriction in software testing," *Inf. Softw. Technol.*, vol. 55, no. 6, pp. 986–1003, 2013.

[P34] Y. H. Tung and S. S. Tseng, "A novel approach to collaborative testing in a crowdsourcing environment," *J. Syst. Softw.*, 2013.

[P35] J. Wang, M. Li, S. Wang, T. Menzies, and Q. Wang, "Images don't lie: Duplicate crowdtesting reports detection with screenshot information," *Inf. Softw. Technol.*, vol. 110, pp. 139–155, 2019.

[P36] S. Alyahya and D. Alrugebh, "Process Improvements for Crowdsourced Software Testing," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 6, 2017.

[P37] S. Guo, R. Chen, H. Li, J. Gao, and Y. Liu, "Crowdsourced Web Application Testing under Real-Time Constraints," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 28, no. 6, pp. 751–779, 2018.

[P38] J. Hernández-González, D. Rodriguez, I. Inza, R. Harrison, and J. A. Lozano, "Learning to classify software defects from crowds: A novel approach," *Appl. Soft Comput. J.,* vol. 62, pp. 579–591, 2018.

[P39] 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 Softw.*, vol. 30, no. 4, pp. 38–45, 2013.

[P40] Q. Cui, S. Wang, J. Wang, Y. Hu, Q. Wang, and M. Li, "Multi-objective crowd worker selection in crowdsourced testing," *in Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, SEKE, 2017, pp. 218–223.

[P41] 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 Proceedings - International Computer Software and Applications Conference*, 2017, pp. 75–84.

[P42] C. Schneider and T. Cheung, "The power of the crowd: Performing usability testing using an on-demand workforce," *in Information Systems Development*, Springer, 2013, pp. 551–560.

[P43] N. Knop and I. Blohm, "Leveraging the internal work force through crowdtesting crowdsourcing in banking," *Int. Conf. Inf. Syst.* 2018, ICIS 2018, 2018.

[P44] N. Leicht, M. Rhyn, and G. Hansbauer, "Can laymen outperform experts? The effects of user expertise and task design in crowdsourced software testing," *24th Eur. Conf. Inf. Syst. ECIS*, 2016.

[P45] N. Leicht, N. Knop, I. Blohm, C. Müller-Bloch, and J. M. Leimeister, "When is crowdsourcing advantageous? The case of crowdsourced software testing," *in 24th European Conference on Information Systems, ECIS*, 2016.

[P46] M. Xie, Q. Wang, G. Yang, and M. Li, "COCOON: Crowdsourced Testing Quality Maximization under Context Coverage Constraint," Proc. - *Int. Symp. Softw. Reliab. Eng. ISSRE*, pp. 316–327, 2017.

[P47] A. Bruun and J. Stage, "New approaches to usability evaluation in software development: Barefoot and crowdsourcing," *J. Syst. Softw.*, vol. 105, pp. 40–53, 2015.

[P48] D. Liu, R. G. Bias, M. Lease, and R. Kuipers, "Crowdsourcing for usability testing," *Proc. ASIST Annu. Meet.*, 2012.

[P49] V. H. M. Gomide, P. A. Valle, J. O. Ferreira, J. R. G. Barbosa, A. F. Da Rocha, and Tmg. Barbosa, "Affective crowdsourcing applied to usability testing," *Int. J. Comput. Sci. Inf. Technol.*, vol. 5, no. 1, pp. 575–579, 2014.

[P50] J. Wang, Q. Cui, Q. Wang, and S. Wang, "Towards Effectively Test Report Classification to Assist Crowdsourced Testing," *Int. Symp. Empir. Softw. Eng. Meas.,* vol. 08–09-September-2016, 2016.

References

[1] M. Alsayyari, S. Alyahya, Supporting Coordination in Crowdsourced Software Testing Services, in: Service-Oriented Syst. Engineering (SOSE), 2018 IEEE Symposium on, 2018, pp. 69–75.
[2] D. Liu, R.G. Bias, M. Lease, R. Kuipers, Crowdsourcing for usability testing, Proc. ASIST Annu. Meet. (2012).
[3] D. Malone, J. Dunne, Social dogfood: a framework to minimise clouc field defects through crowd sourced testing, 2017 28th Irish Signals and Systems Conference, ISSC 2017, 2017.
[4] G. Wu, Y. Cao, W. Chen, J. Wei, H. Zhong, T. Huang, AppCheck: a Crowdsourced Testing Service for Android Applications, in: Proceedings - 2017 IEEE 24th International Conference on Web Services, ICWS 2017, 2017.
[5] R. Musson, J. Richards, D. Fisher, C. Bird, B. Bussone, S. Ganguly, Leveraging the crowd: how 48,000 users helped improve Lync performance, IEEE Softw 30 (4) (2013) 38–45.
[6] S. Alyahya, D. Alrugebh, Process improvements for crowdsourced software testing, Int. J. Adv. Comput. Sci. Appl. 8 (6) (2017).
[7] J. Howe, The rise of crowdsourcing, Wired Mag. (2006).
[8] B. Satzger, Toward collaborative software engineering leveraging the crowd, Econ.-Dri. Softw. Archit. (2014).
[9] N. Leicht, Given enough eyeballs, all bugs are shallow - a literature review for the use of crowdsourcing in software testing, in: Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.
[10] G.J. Myers, T. Badgett, C. Sandler, The Art of Software Testing, 3rd Edition, John Wiley & Sons, 2012.
[11] B. Sterling, The Hacker Crackdown, IndyPublish.com, 2002.
[12] E. Estellés-Arolas, F. González-Ladrón-De-Guevara, Towards an integrated crowdsourcing definition, J. Inf. Sci. (2012).
[13] M. Vuković, Crowdsourcing for enterprises, SERVICES 2009 - 5th 2009 World Congress on Services, 2009.
[14] S. Alyahya, W. Alohali, S. Al-Balhareth, Enhancements for crowdsourced requirements engineering, J. Theor. Appl. Inf. Technol. 96 (12) (2018).
[15] R. Aliady, S. Alyahya, Crowdsourced software design platforms: critical assessment, J. Comput. Sci. 14 (4) (2018).
[16] K.R. Lakhani, D.A. Garvin, E. Lonstein, Topcoder (a): Developing software through crowdsourcing, Harvard Bus. Sch. Gen. Manag. Unit Case (2010) 610–032.
[17] S. Zogaj, U. Bretschneider, J.M. Leimeister, Managing crowdsourced software testing: a case study based insight on the challenges of a crowdsourcing intermediary, J. Bus. Econ. 84 (3) (2014) 375–405.
[18] D.C. Brabham, Crowdsourcing as a model for problem solving: an introduction and cases, Convergence (2008).
[19] M. Hossain, Crowdsourcing: activities, incentives and users' motivations to participate, ICIMTR - International Conference on Innovation, Management and Technology Research, 2012.
[20] E. Schenk, C. Guittard, Crowdsourcing : what can be outsourced to the crowd, and why ? Workshop on Open Source Innovation, 2009.
[21] E. Schenk, C. Guittard, Towards a characterization of crowdsourcing practices, J. Innov. Econ. (2011).
[22] S. Alyahya, M. Alsayyari, Towards better crowdsourced software testing process, Int. J. Coop. Inf. Syst. 29 (1) (2020).
[23] D.C. Brabham, Crowdsourcing, MIT Press Essent. Knowl. Ser. (2013).
[24] L.B. Erickson, I. Petrick, E.M. Trauth, in: Organizational uses of the crowd: developing a framework for the study of crowdsourcing, 2012.
[25] L. Hetmank, Components and functions of crowdsourcing systems – a systematic literature review, Wirtschaftsinformatik (2013).
[26] K.J. Stol, B. Caglayan, B. Fitzgerald, Competition-based crowdsourcing software development: a multi-method study from a customer perspective, IEEE Trans. Softw. Eng. (2019).
[27] R. Gao, Y. Wang, Y. Feng, Z. Chen, W. Eric Wong, Successes, challenges, and rethinking – an industrial investigation on crowdsourced mobile application testing, Empir. Softw. Eng. 24 (2) (2019) 537–561.
[28] T. Zhang, J. Gao, J. Cheng, Crowdsourced testing services for mobile apps, in: Proceedings - 11th IEEE International Symposium on Service-Oriented System Engineering, SOSE 2017, 2017.
[29] K. Mao, L. Capra, M. Harman, Y. Jia, A survey of the use of crowdsourcing in software engineering, J. Syst. Softw. (2017).
[30] A. Sarı, A. Tosun, G.I. Alptekin, A systematic literature review on crowdsourcing in software engineering, J. Syst. Softw. (2019).
[31] B. Kitchenham, S. Charters, *Guidelines for performing systematic literature reviews in software engineering*, Technical Report, Ver. 2.3 EBSE Technical Report. EBSE, Keele University, 2007.
[32] T. Ambreen, N. Ikram, A state-of-the-art of empirical literature of crowdsourcing in computing, in: *IEEE 11th International Conference on Global Software Engineering (ICGSE)*, IEEE, 2016.
[33] N.H. Thuan, P. Antunes, D. Johnstone, Factors influencing the decision to crowdsource: a systematic literature review, Inf. Syst. Front. (2016).
[34] V. Braun, V. Clarke, Using thematic analysis in psychology, Qual. Res. Psychol. (2006).
[35] D.S. Cruzes, T. Dybå, Recommended steps for thematic synthesis in software engineering, International Symposium on Empirical Software Engineering and Measurement, 2011.
[36] I.B. Myers, M.H. McCaulley, N.L. Quenk, A.L. Hammer, MBTI Manual: A Guide to the Development and Use of the Myers-Briggs Type Indicator, Consulting Psychologists Press Palo Alto, CA, 1998.
[37] S. Lazebnik, C. Schmid, J. Ponce, Beyond bags of features: spatial pyramid matching for recognizing natural scene categories, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2006.
[38] R. Chillarege, I.S. Bhandari, J.K. Chaar, M.J. Halliday, B.K. Ray, D.S. Moebus, Orthogonal defect classification—a concept for in-process measurements, IEEE Trans. Softw. Eng. 11 (1992) 943–956.
[39] M. Hosseini, K. Phalp, J. Taylor, R. Ali, The four pillars of crowdsourcing: a reference model, in: Proceedings - International Conference on Research Challenges in Information Science, 2014.
[40] J.A. Konstan, Introduction to recommender systems: algorithms and evaluation, ACM Trans. Inf. Syst. 22 (2004) 1–4.
[41] B. Dit, A. Marcus, Improving the readability of defect reports, Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 2008.
[42] L.L. Cherry, W. Vesterman, Writing tools: The STYLE and DICTION programs, Bill Laboratories (1981).
[43] T. Zimmermann, R. Premraj, N. Bettenburg, S. Just, A. Schröter, C. Weiss, What makes a good bug report? IEEE Trans. Softw. Eng. 36 (5) (2010) 618–643.
[44] S. Herbold, J. Grabowski, S. Waack, U. Bünting, Improved bug reporting and reproduction through non-intrusive GUI usage monitoring and automated replaying, Proceedings - 4th IEEE International Conference on Software Testing, Verification, and Validation Workshops, ICSTW 2011, 2011.
[45] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H.R. Motahari-Nezhad, E. Bertino, S. Dustdar, Quality control in crowdsourcing systems: Issues and directions, IEEE Internet Computing 17 (2) (2013) 76–81.
[46] F. Daniel, P. Kucherbaev, C. Cappiello, B. Benatallah, M. Allahbakhsh, Quality control in crowdsourcing: a survey of quality attributes, assessment techniques, and assurance actions, ACM Comput Surv (2018).
[47] D.E. Difallah, G. Demartini, P. Cudré-Mauroux, Mechanical cheat: spamming schemes and adversarial techniques on crowdsourcing platforms, in: CEUR Workshop Proceedings, 2012.
[48] J. Vuurens, A. de Vries, C. Eickhoff, ', How much spam can you take ? an analysis of crowdsourcing results to increase accuracy, ACM SIGIR Work. Crowdsourcing Inf. Retr. (2011).
[49] X. Zhou, Y. Jin, H. Zhang, S. Li, X. Huang, A map of threats to validity of systematic literature reviews in software engineering, in: Proceedings - Asia-Pacific Software Engineering Conference, APSEC, 2017.