More Testers – The Effect of Crowd Size and Time Restriction in Software Testing

Mika V. Mäntylä[12], Juha Itkonen[2]

[1] *Department of Computer Science and Engineering, Aalto University, Finland*

[2] *Department of Computer Science, Lund University, Sweden*

mika.mantyla@aalto.fi, juha.itkonen@aalto.fi,

Corresponding author: Mika V. Mäntylä,

Publisher's version: http://dx.doi.org/10.1016/j.infsof.2012.12.004

Abstract

Context: The questions of how many individuals and how much time to use for a single testing task are critical in software verification and validation. In software review and usability evaluation contexts, positive effects of using multiple individuals for a task have been found, but software testing has not been studied from this viewpoint.

Objective: We study how adding individuals and imposing time pressure affects the effectiveness and efficiency of manual testing tasks. We applied the group productivity theory from social psychology to characterize the type of software testing tasks.

Method: We conducted an experiment where 130 students performed manual testing under two conditions, one with a time restriction and pressure, i.e., a two-hour fixed slot, and another where the individuals could use as much time as they needed.

Results: We found evidence that manual software testing is an additive task with a ceiling effect, like software reviews and usability inspections. Our results show that a crowd of five time-restricted testers using 10 hours in total detected 71% more defects than a single non-time-restricted tester using 9.9 hours. Furthermore, we use F-score measure from the information retrieval domain to analyze the optimal number of testers in terms of both effectiveness and validity of testing results. We suggest that future studies on verification and validation practices use F-score to provide a more transparent view of the results.

Conclusions: The results seem promising for the time-pressured crowds by indicating that multiple time-pressured individuals deliver superior defect detection effectiveness in comparison to non-time-pressured individuals. However, caution is needed, as the limitations of this study need to be addressed in future works. Finally, we suggest that the size of the crowd used in software testing tasks should be determined based on the share of duplicate and invalid reports produced by the crowd and by the effectiveness of the duplicate handling mechanisms.

Keywords: Software testing, group performance, division of labor, human factors, crowdsourcing, methods for SQA and V&V, Test Management

1. Introduction

The idea of using several people to do the same task is often utilized in the field of software engineering, e.g., in software reviews and usability inspections, and in other fields, such as the academic peer review system. What these tasks have in common is that the purpose is to find defects or perform quality control to improve and asses the item under review. From the effectiveness viewpoint, using several individuals makes sense only if the findings of the individuals differ from each other.

The benefit of using several individuals in verification, validation, and testing has been advocated as a benefit in open-source software development, and it is often referred to as the Linus Law: "Given enough eyeballs, all bugs are shallow" [1]. Based on the software review data of [2], the Linus Law seems to hold true, as adding new individuals to review teams brings up "endlessly" new defects when the number of reviewers exceeds much further than is often considered necessary for software reviews, e.g., even after 20 individuals, adding new reviewers reveals new defects. However, in commercial software development, these open-source techniques cannot be applied as easily, as a higher number of people leads to a higher cost.

Division of labor has scarcely been studied in the area of software testing. With division of labor, we mean whether it is better to have one thorough tester or whether the same total effort is better spent by having several less thorough testers; e.g., assuming that 5 hours of testing effort are available, is it better to have five individuals testing for one hour or one individual testing for five hours? This question is related to both to the effects of individual differences and time restrictions; e.g., if we want to divide a fixed set of effort among multiple individuals and have them perform the same task, we consequently introduce higher time pressure to the equation. We are not aware of any studies in the area of software verification and validation that have researched these questions.

In our prior work [3] the effort distribution problem was formulated as an optimization problem, and the results showed that the optimal choice of the number of individuals depended on the actual defect detection probabilities of the individual defects over time, but also on the size of the effort budget. In [3], we did not have any individuals who worked under time pressure, but we used incomplete review results, i.e. how many defects found after a certain amount of time, to substitute it. This shortcoming is addressed in this paper.

Time pressure has been shown to be beneficial at the project level in software engineering [4], but its effects on smaller software engineering tasks are not known. Yet, many prior works in empirical software testing research have focused on technical approaches, and a recent mapping study regarding test effort reduction showed that 93% of the papers were about technical approaches to testing or defect prediction [5]. At the same time, no research regarding human factors was found. Coincidentally, prior works on testing techniques have found as a *byproduct* that individual differences in software testing are large [6, 7]. The individual differences have been viewed as a drawback, limiting the usefulness and application of testing techniques. However, individual differences can also be viewed as factors that could be utilized with the proper division of labor in software testing.

Another high-profile example of using several individuals in a task can be found in crowdsourcing and beta testing. Since the publication of Surowiecki's popular book on the wisdom of crowds, [8] companies using and studies of crowdsourcing have increased rapidly. Beta testing [9] has been proposed to be the most effective verification and validation measure with a high number of beta-testers (>1,000) [10].

Using a high number of testers is becoming more accessible to even smaller companies, as there currently are online resources, e.g., uTest,[1] where numerous testers can be hired quickly and without cumbersome contractual agreements. Another approach to increase the number of people participating in testing efforts is to use employees with non-testing roles, such as customer consultants and project managers in testing. Using employees with non-testing roles brings the additional benefit of involving people with different viewpoints [11].

In this paper, we study the effects of using several manual testers[2] to test the same functionality from the viewpoint of defect detection performance. First, we create a solid theoretical background based on Steiner's taxonomy of tasks and group performance [12]; study prior works on defect detection performance of individuals and groups, not only in software testing, but also in software reviews and usability; and review the works on the effects of time pressure. Second, we present experimental data collected from 130 students who performed software testing as time-restricted (TR) testers and as non-time-restricted (NTR) testers. Third, we utilize measures originally introduced in the domain of information retrieval (effectiveness, validity, and F-score), which have been used by the software engineering community as well, e.g. [13], but have been, according to our knowledge, ignored in the

[1] www.utest.com
[2] The authors are aware of the academic goals of 100% automated testing [70], but as practitioner reports [71-73] indicate that it is an impossible target, we think that studies on manual testing are also important.

human subject experiments of software engineering. These measures help us understand both the positives and negatives of using multiple testers and allow for the determination of a “sweet spot” in the number of testers.

Next, in Section 2, the theoretical background is presented, Section 3 contains the research methodology, Section 4 shows the results, and Sections 5 and 6 present the discussion and conclusions, respectively.

2. Theoretical background

The use and creation of theory in software engineering research has often been weak, but in recent years, interest has grown among academics and practitioners [14, 15]. The interest of practitioners has been sparked by the observation that, in software engineering, the same things are often re-invented, re-labeled, and re-sold as the latest and greatest development paradigm [14]. This paper supports theoretical development, as we approach software testing through Steiner’s theory of different task types in Section 2.1. Based on Steiner’s taxonomy, we study software verification and validation as an additive task in Section 2.2. Finally, we review the prior work of time pressure and its effect on human performance in Section 2.3, as our study had two conditions, one with time restrictions (TR) and one with no time restriction (NTR).

2.1. Steiner’s taxonomy of tasks

Steiner [12] studied group performance and developed a theory of group productivity. One of Steiner’s key ideas was that studying group performance was meaningless unless one could determine what type task is at hand. We reflect this taxonomy in the contexts of programming and testing in Table 1. However, Steiner’s taxonomy does not work equally for large tasks, e.g., creating a software system, and small tasks, e.g., implementing an individual feature. Thus, we differentiate in terms of task size. In Table 1, we have two types of tasks, large, requiring a team to be implemented in a reasonable period, and small, which could be implemented by a single individual in a reasonable period. The most significant prior work utilizing Steiner’s taxonomy in the context of software engineering was done by Balijepally et al. [16], who compared solo and pair-programming in the context of small tasks with a student experiment. Their results for the small programming tasks are presented in Table 1. In this paper, we continue the work of Balijepally et al. [16] by studying small testing tasks.

According to Steiner, a task can vary in three dimensions. The first dimension considers whether the task is divisible (can be divided into sub-tasks and distributed to separate individuals) or unitary (cannot be divided into subtasks and cannot be feasibly be distributed to separate individuals). For example, painting a house is a divisible task, while reading a book is unitary. In software engineering, tasks can be broken down into subtasks when the task is big enough; e.g., performing system testing for a whole software system is divisible, whereas testing an individual feature is not. In Table 1, we consider that both large testing and programming tasks are divisible, while small testing and programming tasks are unitary.

Steiner's second dimension considers whether the task is maximizing or optimizing. In maximizing tasks, the goal is to get as many items as possible, whereas in optimizing tasks, the goal is to get a single item with the highest possible quality, e.g., creating a high number of new ideas vs. creating the best idea. We think that large tasks for both programming and testing have both maximizing and optimizing elements. However, for small tasks, i.e., programming or testing of an individual feature, there is a difference. We think that programming tasks are optimizing, e.g., coming up with the best possible way to implement a feature (also shown in [16]), but that testing tasks are maximizing, e.g., revealing as high a number of relevant defects affecting the feature as possible.

The third dimension in Steiner's taxonomy is the combinability of the tasks, meaning how the efforts of a group performing the task can be combined. The combinability can be additive, conjunctive, or disjunctive. In additive tasks, the sum of all performers is considered, e.g., pushing a car, and the group performance is the sum of individual contributions. In conjunctive tasks, every group member must perform; e.g., all members of a mountain climbing team must climb, and the team is as slow as its weakest member. In disjunctive tasks, it is enough that one group member is able to perform the task, e.g., figuring out the correct answer in math assignment, and the team performance is determined by the best-performing member. Again, for software engineering, the size of the task matters, as large tasks (which are divisible) can always be considered non-disjunctive. Depending on the setup, they can be additive (the sum of all performers determines the output), conjunctive (the weakest performer determines the output), or both. An example of a conjunctive large task can be a case where the overall quality of software is determined by the module with the lowest quality that has been created and tested by the weakest members. When considering small tasks, programming tasks become disjunctive (as shown in [16]), and only the output of the best performer determines the output in pair-programming tasks. The focus of this paper is on small testing tasks, and in this paper, we claim that they are additive based our results and discussion in Sections 4 and 5 and prior work in Section 2.2.

Finally, we would like to note that Steiner's taxonomy of tasks is a simplification of the real-world context and that often, in reality, the type of a task is on a continuous scale between the alternatives, e.g., between maximizing and optimizing. Economic realities can transform programming tasks from optimizing to maximizing. When the goal is to complete a programming task as fast possible to maximize revenue with an acceptable level of quality, the task would be about maximizing revenue rather than optimizing on quality. Nevertheless, Steiner's taxonomy works as a theoretical frame of reference that allows us to make meaningful comparison with two core software engineering tasks, programming and software testing.

Table 1. Steiner's taxonomy and software engineering tasks

Dimensions	**Large tasks**		**Small tasks**	
	Programming	**Testing**	**Programming** [16]	**Testing**
Divisible-Unitary	Divisible	Divisible	Unitary	Unitary
Maximizing-Optimizing	Both	Both	Optimizing	Maximizing
Additive-Conjunctive-Disjunctive	Non-Disjunctive	Non-Disjunctive	Disjunctive	Additive

2.2. Software verification and validation as an additive task

In this section, we will look at the background that makes software verification and validation an additive task according to Steiner's taxonomy of tasks [12]. In addition to testing, we review relevant prior works from the domains of software reviews and usability inspections since we think that software reviews and usability inspections are also additive tasks. In Section 2.2.1., we review prior works focusing on human cognition that can help us understand the reasons for the additive nature of software testing. In Section 2.2.2., we review the defect detection effectiveness of groups and, in Section 2.2.3., the defect detection effectiveness of individuals in prior works, and finally, in Section 2.2.4, we provide a brief summary of the topic.

2.2.1. Human cognition

Many cognitive biases[3] and traits affect humans performing verification and validation. Such biases will lead to missing defects and to detecting false positives. These biases can help us to understand the roots of the additive nature of software testing. These biases are under increasing interest, for example, in the medical domain where incorrect diagnoses caused by cognitive errors have been studied [17, 18] and countered by the practice of having a mandatory second opinion [19].

An article about positive test strategy [20] showed that humans are four times more likely to create positive test cases, i.e., cases confirming their hypothesis, than negative ones. Thus, having only a single individual to test the system offers a limited viewpoint, as the individual is likely to stick with his or her original hypothesis. If, for

[3] http://en.wikipedia.org/wiki/List_of_cognitive_biases

example, four individuals test the system, then the system is tested starting from four possibly different initial hypotheses rather than just one; thus, the likelihood of finding errors is far greater.

In the area of usability testing, it has been found that people with a certain cognitive style, field independent [21], are significantly better at finding usability problems than those without this trait [22]. On the other hand, in [23] the researchers studied software reviews and found that cognitive styles measured with Myers-Briggs type indicator did not explain the variation between individuals and, furthermore, that it did not help in creating optimal defect detection teams, either. Thus, the evidence linking cognitive differences between defect detection effectiveness are conflicting and scarce.

The limited attention of an individual is another factor explaining why more individuals are better at software testing. A well-known example of this is a website[4] where people are asked to count the number of times a ball is thrown in a video. Afterwards they are asked not how many times the ball was thrown but whether they saw the gorilla in the video. The results show that the gorilla is missed by over half of the people watching the video, as they are too focused on the main task that is given to them [24]. This phenomenon is referred to as inattentional blindness [25], and it has been shown in numerous experiments. Thus, adding more people watching the gorilla video increases the probability of seeing the gorilla. We can also easily understand that, when some defects in software testing are missed due to limited attention, adding more people to the task is likely to increase the detection probability.

Recently, Fry and Weimer [26] studied human fault localization accuracy in code reviews and found that humans much more frequently observe certain types of defects than others. For example, extra code statements are recognized over five times more frequently than missing statements. Furthermore, in certain programs, errors were much easier to detect than in others, e.g., humans were over three times more accurate in detecting errors in programs using arrays than in programs using a tree structure. This study indicates that, for particular defects, using more individuals may be more required than in others.

Other factor affecting defect detection is knowledge or expertise. In the usability domain, it has been well-demonstrated that individuals with greater expertise find more usability issues than individuals with lower expertise. However, we also find that lacking expertise can be compensated to a certain degree with having more individuals with lesser expertise. For example, three novices together find the same number of usability problems as one

[4] http://www.theinvisiblegorilla.com

usability specialist [27]. However, even for expert individuals, i.e., usability specialists, the additive nature of usability inspections persisted. Recently, we studied [28] with a think-aloud protocol how knowledge is used as a test oracle in software testing. The testers used their knowledge of the application domain and the system under test when reasoning whether a particular behavior or result is a defect or not. In the study, numerous defects would have not been detected if the tester had not possessed high system or domain knowledge.

2.2.2. Defect detection effectiveness of groups

Several experiments have been conducted on manual testing comparing different testing techniques, and it seems that there are no differences in terms of the number of defects detected between testing techniques on average but that different techniques detect somewhat different defect types and that different techniques are effective in different programs [7, 29-31]. Thus, it has been concluded that a combination of different testing techniques will lead to the highest defect detection effectiveness. However, a topic that has not been well-communicated is that higher defect detection effectiveness in combining different techniques in manual testing is mostly produced by having more testers rather than more techniques. Wood et al. [7] provided data showing that a combination of two testers with different testing techniques yielded an average defect detection effectiveness of 74.8%, but a combination of two testers with the same technique led to an average defect detection effectiveness of 72.1%. Thus, the improvement from using two different testing techniques is 2.7 percentage points (an increase of 3.7%) in detecting defects. However, a combination of any two testers led to a detection effectiveness of 73.0%, while the average defect detection effectiveness of a single tester was only 56.5%. Thus, the improvement from using two testers instead of one is 16.5 percentage points (an increase of 29.2%). This provides an initial indication that software testing is an additive task.

Biffle and Halling [2] studied the impact of process mix and team size on inspection performance with nominal teams. They found that the process mix was more important than team size for five or more inspectors and that the difference was statistically significant for eight or more inspectors ($p=0.1$). In other words, for teams of five or more inspectors, it was more important to have a process mix than to add additional inspectors, and for smaller teams, it was vice versa. What can also be seen from their data charts but is not a part of their research questions is that it was slightly more beneficial to have a single inspector read for four hours (18% of defects) than to have two inspectors read for two hours (16% of defects). However, it was slightly less beneficial to have a single inspector read for six hours (21% of defects) than to have three inspectors read for two hours (22.5% of the defects). Thus, their results indicate that it does not matter how one chooses to distribute effort among reviewers. However, we believe that their experimental setting does not allow us to make such conclusions. This is because no time pressure was present in their study; i.e., all participants were allowed to read as long as they liked. The different time groups were created artificially by counting how many defects each individual had found after, e.g., two hours. We have good reasons to believe that, if real time pressure had been present, the results would have been different (see Section 2.3.).

In the area of usability inspections (also known as user testing), having several inspectors has been proposed to find more usability issues and to reduce the evaluator effect, i.e., different evaluators detecting a different set of problems [32]. Fölstad [33] studied the usability inspection performance between work-domain experts and usability experts and found that usability experts found, on average, 40% of the usability problems, while work-domain experts found 25%, on average. However, when the number of evaluators was increased to five, the work-domain experts detected 55% of the problems, while the usability experts detected 68% of the problems. Similar results have been provided by others. For example, Nielsen [27] showed that a single evaluator found, on average, less than 50% of the problems and that the average for four evaluators was over 80%, and Sears [34] found that the percentage of problems found increased from 41% to 61% when the number of evaluators was increased from 2 to 5. However, in none of those studies was the effect of time pressure present.

Finally, we need to recognize work on capture-recapture models where the goal is to estimate the defect content of an artifact with different mathematical models based on the shared and unique defects found by the individuals [35]. A key lesson from capture-recapture is that defects have different detection probabilities and that many individuals are needed to get reliable estimates [36, 37]. We see that work on capture-recapture models supports the theory that software reviews are an additive task (see Table 1).

2.2.3. Defect detection effectiveness of individuals

The individual's defect detection effectiveness is an important factor affecting whether using several people yields to benefits. The defect detection effectiveness[5] of individuals in human-based verification and validation; i.e., in manual software testing, software reviews, and usability inspection, varies considerably between different data sets. Hertzum and Jacobsen [32] showed that, in usability inspection, it varies between 18% and 73% (average 37%, median 33%), and further studies have produced similar values: 33% in [33] and 12% in [22]. In software reviews of software requirements documents, an average defect detection effectiveness of 9% to 21% was observed in [2], in

[5] The detected percentage of total or known defects

[38], the average was 7% to 16%, and in [37], the average was 10%. In code reviews, the average number of defects detected was found to be between 29% and 66% in [39], and in [26], the average was 46%. Finally, in software testing experiments with small programs, the averages of 46% in [7], 48% in [30], 36%-48% in [40], and 42% in [41] have been reported. As a general trend, it seems that numbers tend to fall below 50%; very low numbers (under 10%) can be seen, while large percentages indicating over 90% defect detection effectiveness are missing (73% is the highest average reported). This suggests that using a single individual to perform verification and validation alone is a not suitable approach in many cases.

Several factors affect an individual's defect detection effectiveness, and they can be partly explained by human cognition (see Section 2.2.1). For the purposes of our study, one of the most important ones is the speed of the verification and validation task. Kemerer and Paulk [39] studied how the review rate, i.e., the speed of code reading, affected the number of defects detected by individual reviewers. They found that the highest defect removal effectiveness was achieved with the slowest review rate: less than 200 LOC/h, leading to a detection rate of 56% to 66%. They also found that detection effectiveness declined as the review rate increased and that reviewers going at a speed of over 800 LOC/h found only 29% to 33% of the defects. Although a higher review rate leads to decreased defect detection effectiveness, we cannot claim that faster speed would be a worse solution, as it is associated with less effort. Furthermore, based on [39], we cannot assess whether one reviewer going at a rate of 200 LOC/h is better than having four reviewers going at a rate of 800 LOC/h.

2.2.4. Summary

In Subsection 2.2.1, we presented prior work on human cognition that can help us to understand why software verification and validation tasks are additive. However, because human cognition is far from being completely understood, we could only offer partial hints to this question.

We reviewed prior work on group performance in software verification and validation in Subsection 2.2.2. The results come mostly from usability inspections and software reviews and show that adding more individuals to verification and validation task leads to higher defect detection effectiveness. This means that verification and validation seems to be an additive task. Furthermore, concerning software testing, we were able to find only one study of the matter, and that study only showed data for combining two individuals, and the testing was performed in an automated unit-testing context with a program of few a hundred lines; the study was quite different from our study, which is explained in Section 3.2.

Finally, in Subsection 2.2.3, we reviewed studies presenting the defect detection effectiveness of individuals. In many of those studies, individual defect detection effectiveness was below 50%, and in several examples, the detection effectiveness was less than 20%. This is important since, when individual defect detection effectiveness is low, it means that a substantial contribution can be made by adding another individual to the V&V task. For example, if a single tester would find an average of 90% of the defects, then using more testers could offer only limited benefits.

2.3. Effects of time pressure on task performance

Several studies have assessed time pressure and its effects on software development at the project level with real [4, 42] and simulated data [43, 44]. These works differ from this study, as they assess the project context, i.e., large tasks in Table 1, whereas we are focusing on small tasks. In [4], time pressure was seen to have a U-shaped effect, called the Yerkes-Dodson law [45], where a small amount of time pressure improves performance, but when time pressure becomes excessive, it has negative effects due to increased errors and poor individual commitment due to unrealistic schedules. We think that studies of small tasks are also important because they allow better control and more accurate measurement than large tasks at the project level.

In the small task context, the only study of time pressure in the software engineering domain that we know of is by Topi et al. [46]. Their work on student subjects on creating database queries showed that time pressure had no effect, and individual performance was explained by the skills of the subject, task complexity, and availability of time. However, in comparison to our work, there are significant differences. First, their time limits and tasks were significantly smaller than ours, lasting between 4 and 21 minutes. Second, they had no condition where time pressure was not present, as not all of their subjects were able to complete the tasks even in the condition with the greatest amount of time. Third, their task was comparable to a small programming task (see Table 1). Thus, it was disjunctive and optimizing, while ours was additive and maximizing.

Another relevant small task study focusing on time pressure comes from the domain of accounting. McDaniel [47] performed an experiment with 179 professional staff auditors and showed that time pressure and time restriction increased efficiency but decreased the effectiveness of individual auditors. However, the study gave no hints as to whether, with equal total effort, it is better to have several time-restricted auditors than a single non-time restricted auditor.

3. Method

3.1. Research goals and questions

The research goal of this paper is *to study manual software testing to understand how division of labor should be performed*. In particular, we are interested in the following research questions:

- RQ1: What is the effect on the defect detection performance of adding individual testers to a non-communicating crowd of testers?
- RQ2: What is the effect of time restriction on the defect detection performance between TR and NTR testers?

  a) On an individual level

  b) On a crowd level when controlling for effort

In RQ1, we assess what the added benefits and drawbacks are of having several individuals perform manual system level testing of the same functionality. In RQ2, we study the effect of a time restriction on individual performance and how several TR individuals compare with an NTR individual when controlling for total effort. In other words, RQ2 tries to determine which one is the better division of labor: having a single thorough tester (NTR) or many less thorough testers working under a time restriction (TR).

We have chosen to use the word “crowd” because, according to [8], wise crowds are characterized by independence and decentralization, which matches our setup. We also think that the words “group” and “team” suggest, in the SE context, that work is coordinated and done together, instead of simply combining individual effort without coordination. Past work has used terms “virtual team” or “nominal group”, but those terms are also problematic. The term “virtual team” can additionally refer to a real team that works in a distributed environment, thus creating a semantic mismatch with our setup. Furthermore, the term nominal group is unfamiliar to many software engineering practitioners and even scholars, according to our experience.

In the existing works, often, only positive effects have been analyzed, i.e., the number of detected defects. In this paper, we also consider negative aspects in terms of the number of invalid defect reports and aspects that can be considered either negative or positive in terms of the number of duplicate defect reports. Unique defects are clearly a benefit; a larger number indicates better performance in testing. Invalid defect reports indicate a defect that does not really exist or a defect report that is incomprehensible. Invalid defect reports take effort to create, and they must be studied and processed, but they give no benefits in terms of providing information about the product quality. The number of invalid defect reports is clearly a cost; larger numbers indicate poorer performance. Finally, the number of duplicate defect reports can be seen either as a benefit or as a cost. Traditional wisdom regarding duplicate defect reports consider them only a cost because they give no information about the product quality in terms of new, unique defects, and they still require effort in defect reporting and processing. However, they can be valuable in at least three ways. First, they can be used to verify that a certain defect really exists in the system and that a defect is replicable. Second, they might provide information about the likelihood that someone will actually encounter this defect in real use. This can be useful in prioritizing what defects should be fixed based on the number of individuals reporting the defect. Third, developers appreciate the amount of additional information they bring [48]. Due to this conflicting view of duplicate reports, in our analysis, we consider duplicate reports in two ways, first as harmful (duplicates as invalid defect reports) and second as something that cannot be considered positive or negative (duplicates as ignored). The latter approach is adopted from usability inspection practice. We should also consider duplicate as positive, but we see this as a complex task and something that should be done in future work rather than in this article.

3.2. Subjects

The experiment was performed in sequential years in the software testing and quality assurance course at Aalto University, Helsinki, Finland. In the first year, 79 students participated in the experiment under the TR condition, and in the second year, 51 students who performed testing under the NTR condition. The different number of students in subsequent years simply reflects the fluctuation of the popularity of the course, the number of CS students majoring in software engineering, and other variations in the university's educational programs. Table 2 shows the student demographics of the TR and NTR groups collected using a survey form. To understand the demographic data, we need give some information about our university system. At our university, the students have no tuition fees, and they can progress at their own pace in their studies. Thus, it is possible, for example, that a third-year student and a ninth-year student have the same number of credits. However, often, the ninth-year student would have several years of work experience in software development due to high number of IT companies in our university's region. Actually, according to our students, working in the industry is one of the top reasons that many students fail to complete their studies in the five-year period as instructed but not enforced at our university.

To capture the variation among our student population, we surveyed them regarding their study year, credits, and professional testing and programming experience. After the course, we also collected the grade they received from the course. In Table 2, we can see that the NTR group was slightly more advanced when measured by the number of credits and the percentage of people having professional testing and programming experience. The TR group was more senior in years of studies and amount of professional programming and testing experience for the individuals who had professional experience. The TR group also had a slightly higher percentage of people who passed the course. To summarize the differences between student populations, we state that, first, the differences are small, and, second, while some measures favor the TR group, other measures favor the NTR group. We tested the differences between groups with a t-test for credits and with a Mann-Whitney test for other variables (level p=0.05). Note: Because we tested six variables, our chance of finding significant differences is actually 26.5%. We found only one significant, but small, difference; the NTR group had higher mean testing experience. The mean testing experience for all participants was 0.44 years in the TR group and 0.58 years in the NTR group. In other words, the level of testing experience was low in both groups. Thus, based on the demographic data, we think that there were, in fact, no such differences between the populations used in the experiment that would largely affect the results.

Table 2. Demographics of the student groups used in the experiment

	TR (n=79)	NTR (n=51)
Credits: average/median/st. dev.	113/110/36.8	124/120/33.9
Years of study: average/median/st. dev.	4.75/4/1.77	4.58/4/1.46
% of individuals who passed the course	94%	88%
Passing grade (1-5) average/median/st. dev.	3.48/4/1.1	3.5/4/1.4
% of individuals with professional programming experience	63%	71%
Years of professional programming experience[1] average/median/st. dev.	2.92/2/2.77	2.01/2/1.38
% of individuals with professional testing experience	21%	43%
Years of professional testing experience[1] average/median/st. dev.	2.07/1.5/1.36	1.36/1/1.19

[1] Individuals with zero years of experience were excluded from calculations

Using students in software engineering research is acceptable if they are trained and used to establish a trend [49]. Both conditions are met in our experiment. All the students used as subjects were participating in a software testing course where they were taught software testing fundamentals before the experiment. Furthermore, students can be considered representative of beginner testers in the industry. Furthermore, Eldh et al. [50] showed that many industrial testers do not, in fact, know how to apply some basic testing techniques. Thus, it is not only students who lack knowledge of software testing techniques. In addition, another prior experiment showed no differences between students and professionals [51].

3.3. Experimental setup

See Figure 1 for an illustration of the experimental setup that was used in subsequent years. In the first year, the students performed time-restricted testing in two sessions, each lasting 2 hours with two feature sets of the jEdit text editor, denoted feature set A and B in this paper. In the second year, students tested only the feature set B, and they were allowed to use as much time as they needed. Only one feature set was tested in the second year, as we wanted to create an NTR condition and did not want to make large increases in the total effort of the students. On average, the students used 9.87 hours in the second year to test feature set B, roughly five times more than they used in the first year. In the first year, the testing was performed in a computer classroom where the course staff was present. In the second year, the testing was performed as an individual exercise without controlled sessions, and the students were required to report the number of hours they used, but the hours used had no effect on their grade. In both years, the students' performance was evaluated based on the number of defects (55% of the exercise grade) and the quality of the defect reports and the test logs (45% of the exercise grade). The grading principles were announced to the students before the exercise. Therefore, we believe that, in both years, the students were motivated to try their best. The data for the experiment was collected from an experiment where a comparison was made between exploratory and test case-based testing [31]. The results showed that there were no significant differences between the techniques regarding the total number of defects or different defect types. The only significant difference found was that test case-based testing produced more invalid defects. In the second year, the participants also used both exploratory testing and test case-based testing, but only for two subsets of feature set B. In both years, the same testing techniques were used by all students, and there were no significant differences regarding defect detection effectiveness; thus, in this paper, we have merged the dataset between the techniques. This paper focuses on the differences between different-sized nominal testing crowds and the differences between the time-restricted (TR) testers of the first year and the non-time-restricted (NTR) testers of the second year. Finally, we think that the students in the TR condition were truly under time pressure as, for feature set A, 39% percent of students complained about the lack of time in the open survey questions, and for feature set B, it was 29%. Since these numbers come from open questions, where students typically wrote only two statements or less, we have a reason to believe that, in fact, a greater number of students suffered from time pressure. Furthermore, in the second year, the minimum time reported by 51 NTR testers was 2.75 hours. Therefore, we have good reason to believe that the time restriction in the first year was real and significant.

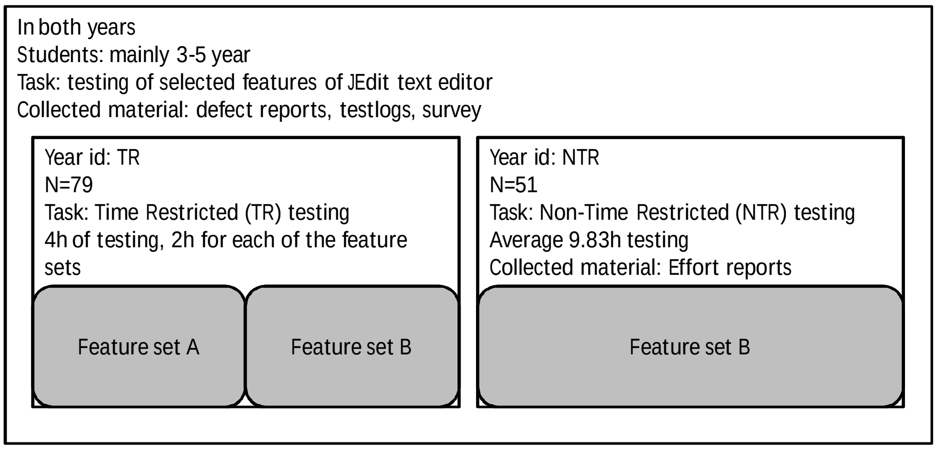
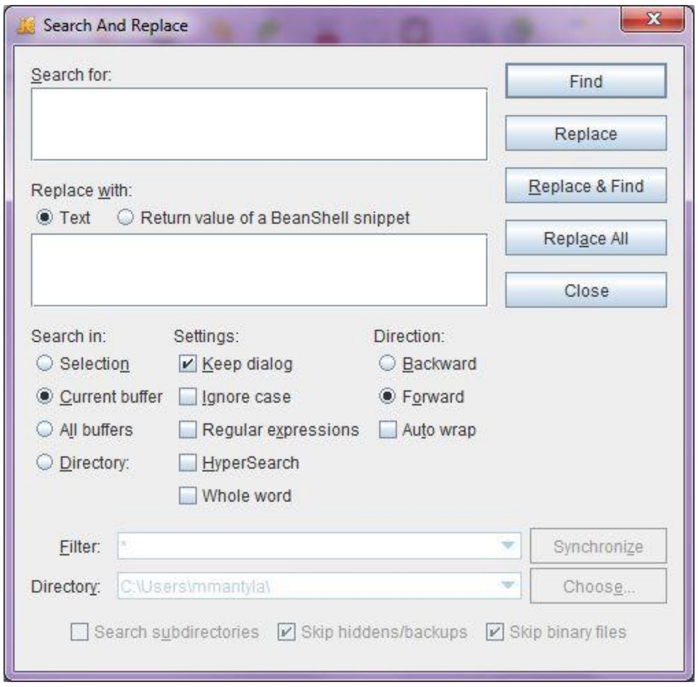

In both years
Students: mainly 3-5 year
Task: testing of selected features of JEdit text editor
Collected material: defect reports, testlogs, survey
Year id: TR
N=79
Task: Time Restricted (TR) testing
4h of testing, 2h for each of the feature sets
Feature set A
Feature set B
Year id: NTR
N=51
Task: Non-Time Restricted (NTR) testing
Average 9.83h testing
Collected material: Effort reports
Feature set B

Figure 1. Experimental setup

The software being tested was jEdit text editor version 4.2, and it was tested through a graphical user interface (see Figure 2). We used a version of jEdit that we had seeded with defects; the feature sets A and B had 25 and 24 defects, respectively. In addition, the software contained real defects that were revealed during the experiment. In both years, the same version of jEdit was used. JEdit versions with seeded defects were not available to the subjects before the test sessions. The normal open-source version of the software was available to the subjects beforehand so they could familiarize themselves with the features. The JEdit text editor was selected for testing because the domain knowledge for understanding the basic functionalities of a text editor was well-possessed by the students through the prior use of jEdit or similar text and code editing software.



Search for:

Replace with: [✓ Text] Return value of a BeanShell snippet

Search in: Selection [✓ Current buffer] All buffers Directory:

Settings: [✓ Keep dialog] Ignore case Regular expressions HyperSearch Whole word

Direction: Backward [✓ Forward] Auto wrap

Filter: *

Directory: C:\Users\mmantyla\

Search subdirectories [✓ Skip hiddens/backups] [✓ Skip binary files]

Figure 2. Screenshot of the jEdit text editor. The "Search And Replace" functionality was part of feature set B.

3.4. Data analysis

Students submitted defect reports after they had completed testing. The second author analyzed each defect description and assigned a corresponding defect ID to it. To study the performance of using several testers, we sampled the order in which testers are assigned to a group and analyzed how many new unique defects would be found by each added tester. We did this by creating 10,000 random permutations of how the testers could be ordered. We performed the randomization, sampling, and calculations using an R statistical analysis and a programming environment. We chose 10,000 permutations, as with test runs with 1,000 permutation runs, we found that differences between runs affected the numbers in the second decimal; thus, we figured that the likelihood of any significant random variation would be mitigated with 10,000 permutations. Furthermore, the time for running 10,000 permutations was suitable and not too extensive, i.e., roughly 5 hours. From our calculations based on random permutations, it is possible to see the average number of unique defects found by various crowd sizes. We also analyzed distributions, quartiles, and extreme cases of minimum and maximum numbers of unique defects. We did the simulation because creating all permutations would have been computationally exhaustive. We found that, from the group of 79 testers, we could create 3,160 unique combinations with two testers, but with three, four, and five testers, the number of combinations is 82,160, 1,581,580, and 24,040,016, respectively.

For comparison between groups, we report effect sizes using Cohen's *d* [52, 53], rather than statistical significances. Statistical tests indicate the likelihood of finding the observed result by chance, whereas the effect size reports the practical significance as it measures the magnitude of the treatment effect. Furthermore, one could have a high statistical significance and a low effect size simultaneously by having a high number of subjects and a treatment that produces only a small effect. The reverse, i.e., having a low statistical significance and a high effect size, can be observed if one has a small number of subjects, which does not allow the high effect size to become statistically significant. Relying on statistical tests that are based on a sample size is problematic in our case. As we have simulated data for the combinations of testers, large numbers of data points are randomly generated from the same primary data form 79 actual testers. Since the data points are not a random sample from any real population, the statistical tests based on the sample size are not meaningful. Our sample sizes would be very large for the simulated results and would, therefore, easily provide statistically significant results, even with small differences between the groups. Thus, we think that reporting effect sizes directly yields a more honest and meaningful comparison.

Cohen's *d* is the difference of means between groups divided by the standard deviation [52, 53], and it assumes a normal distribution. The standard deviation can be either the control group or a pooled standard deviation. We used the control group (NTR) standard deviation, as it had higher standard deviation in all cases, which lead to smaller effect sizes and to erring on the side of caution. The assumption of normal distribution may not always hold for software engineering data, and the confounding effect of sample size makes normality testing questionable (see footnote[6]); thus, we also computed unbiased Cohen's *d* as described by Kelley [54]. In all cases, the unbiased Cohen's *d* provided larger effect sizes on average[7] than we present in this paper, as Kelley's method used the pooled standard deviation. The bottom line is that the method we used for analyzing Cohen's *d* makes no difference to the conclusions or to the high-level results of this paper.

According to Cohen [55], the *d* value of 0.8 indicates a large effect, 0.5 indicates a medium effect, and 0.2 indicates a small effect. More detailed description of *d* value [55][8] states that value 0.8 indicates that the treatment

[6] http://blog.fellstat.com/?p=61

[7] The Kelley's method [54] for computing unbiased *d* used R's gamma function that only allowed 170 inputs so we looped over our 10,000 simulation results and picked the average.

[8] For online reference of the interpretation of *d*, see http://www.umdnj.edu/idsweb/shared/effect_size.htm

group mean stands at the 79th percentile of the control group, *d* 0.5 yields to the 69th percentile, and finally, *d* 0.2 yields to the 58th percentile. If the mean of the treatment group stands at the 95th percentile of the control group, then this leads to *d* of roughly 1.65. Cohen's *d* can also have negative values, which are interpreted in the same way as the positive values but with a reversed effect. In this paper, negative values indicate that the NTR group has a higher mean, while positive values indicate that the TR group has a higher mean.

3.5. Measures

We use measures that were originally introduced in the information retrieval domain [56] but have since been partially adopted by the usability community [57] and the software engineering community. These measures allow the measuring and depicting of both the negative and positive aspects of using multiple testers.

The share of unique defect findings among all the known defects in the software allows a comparison between different data sets. The measure of the share of unique defects found by a group of testers is called *effectiveness*, as we want to make this paper backward-compatible with the works of software reviews and testing we reviewed in Section 2.2 (in the usability domain, this measure is called thoroughness Hartson et al. [57], and in the information retrieval domain, it is recall [56]). Effectiveness is defined as follows:

$$E = \frac{vdf}{vdt}$$

where *E* is effectiveness, *vdf* is the number of unique valid defects found, and *vdt* is the number of total unique valid defects.

Although each tester will find a certain share of unique defects, they additionally produce a set of invalid defect reports (also referred to as false positives [58]). The share of valid unique findings among all findings is called *validity* [57] (in the domain of information retrieval, this is called precision [56], and in empirical software engineering, this number is often not reported; see the papers reviewed in Section 2.2). In this paper, we define validity as follows:

$$V = \frac{vdf}{vdf + idf}$$

where *V* is validity, *vdf* is the number of unique valid defects found, and *idf* is the number of invalid defects found.

Finally, it is useful for decision-making purposes to provide a combination measure of effectiveness and validity. The importance of this combination measure is in determining the best number of testers because, in practice, increasing the number of testers always results in an increase in effectiveness and a decrease in validity, as we shall see in Section 4. Hartson et al. [57], in the usability domain, defined a measure for this as a product of effectiveness and validity. In the information retrieval domain, both effectiveness and validity are combined in a measure called the F-score, which is the harmonic mean of effectiveness and validity [59]. The F-score is better, as it gives values that are more comparable to effectiveness and validity, as we can see in Table 3. For example, when both effectiveness and validity are 0.5, the combination measure should also be 0.5. This is true for the F-score but not for the effectiveness measure by Hartson et al., which gives a value of 0.25 (see Table 3 for examples).

Table 3. Comparing the combination measure by Hartson et al. [57] **and the F-score** [59]

Effectiveness	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.10	0.20	0.30	0.40
Validity	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.90	0.80	0.70	0.60
Hartson et al. [57]	0.01	0.04	0.09	0.16	0.25	0.36	0.49	0.64	0.81	0.09	0.16	0.21	0.24
F-score [59]	0.10	0.20	0.30	0.40	0.50	0.60	0.70	0.80	0.90	0.18	0.32	0.42	0.48

Thus, we use the F-score as the effectiveness measure, as it is mathematically more solid, and we calculate it using the effectiveness and validity measures as follows:

$$F = \frac{2 \cdot E \cdot V}{E + V}$$

In this paper we only report F_1-score that puts equal weight on effectiveness and validity. During the data-analysis, we also used F_β-score; $F_{0.5}$-score puts more weight on validity, and F_2-score puts more weight on effectiveness. However, this did not affect whether one should choose TR or NTR testers and, thus, we do not report these results in this paper. Naturally, $F_{0.5}$-score favored a smaller number of testers and F_2-score favored a higher number of testers than F_1-score, but this follows straight from the mathematical equations and the fact that effectiveness always increases and validity decreases when more testers are added to the crowd.

4. Results

In this section, we present the results. First, we study effectiveness in terms of the number of unique defects with different sizes of non-communicating crowds of testers in Section 4.1. Then, in Section 4.2, we study the validity and F-score as defined in Section 3.5.

4.1. Number of defects and effectiveness

Table 4, Figure 3, and Figure 4 show the number of unique defects found in different conditions, and Figure 5 shows the effectiveness, i.e., the percentage of the total unique defects found. In Table 4, the leftmost column indicates the number of testers, and the other columns show the number of unique defects found by each number of testers in the three conditions: A TR, B TR, and B NTR. In the four rightmost columns in Table 4, we provide an easy comparison between the testers with time restriction (B TR) and with no time restriction (B NTR). Since the NTR testers used 9.87h, on average, while the TR testers used only 2h, it is sensible to compare how one NTR tester (effort of 9.87h) compares against 2-5 TR testers (total effort of 4-10h). The column headings 2-5 * B TR indicate the number of TR testers used in the comparison. For example, if we wish to compare how one NTR tester compares against five TR testers, we would choose the first row under the column headings of B NTR and 5 * B TR, which would give us values of 11.28 and 19.32, and if we want to know how two NTR tester compare against 10 TR testers, we would choose the same columns headings but go to the second row and get the values of 16.98 and 25.77. Furthermore, in Table 4, we can see, in addition to the average, the 5th and 95th percentiles, which indicate the variation within each condition.

Table 4 and Figure 4 show that the number of unique defects increases as the number of testers increases. It is better to have n+1 testers than only n testers in terms of the number of unique defects found in all of our data sets. Naturally, the benefits of adding more testers diminish as the number of testers increases, which is due to the exhaustion of unique defects. However, in all conditions, it appears that adding testers up to very high numbers increases the number of unique defects, as shown in Figure 4. For example, after 40 testers, adding an additional tester from the same population adds 0.3 new defects in the TR condition and 0.15 new defects in the NTR condition. This is due to the fact that many of the defects are found by only one or two individuals. Thus, having a number of testers that seems too large for traditional software engineering, e.g., 40, can actually be a good quality assurance practice if a very high level of quality is required.

We found that time restriction has a negative effect on the defect detection effectiveness at the individual level. Table 4 shows that a single NTR tester finds 11.28 defects, on average, and a single TR tester finds 7.58 defects, on average. The effect size of this difference in Cohen's *d* is -0.95, indicating a large effect (Note: the tables of this paper do not explicitly state the *d* values). However, time restriction has a positive effect on efficiency, as TR testers find 3.79 defects/h, while NTR testers find only 1.14 defects/h.

Furthermore, time restriction has a positive effect on defect detection effectiveness when comparing effort-wise equal time-restricted crowds against individuals without time restrictions. A single NTR tester uses 9.87 hours to find 11.28 defects, while five TR testers use 10 hours to find 19.32 defects, which is, effort-wise, the closest match between the TR groups and individual NTR testers in our data. The effect size in defect detection effectiveness between these two crowds in Cohen's *d is* 2.06, indicating a very large effect. When using crowds, additional effort is required to coordinate the work and to manage and merge the individual contributions of multiple testers. Therefore, we have calculated the effect sizes of using four, three, or two TR testers, who find, on average, 17.44, 15.11, and 12.07 defects and use 8, 6, and 4 hours, respectively. The effect sizes of defect detection effectiveness in comparison to a single NTR tester are 1.58, 0.98, and 0.20.

Finally, having more testers makes the fluctuation in defect detection effectiveness smaller, as seen when looking at the 5th to the 95th percentiles in Table 4. For example, a single NTR tester and two TR testers find 11.28 and 12.07 defects, respectively, on average. However, the 5th to the 95th percentile ranges are smaller for two TR testers (4-18 vs. 8-17 defects). The relative difference between the 5th and the 95th percentile decreases in all conditions when we have more testers, e.g., with 2 NTR testers, the 95th percentile finds 109% more defects than the 5th percentile (23/11), but with 10 NTR testers, the 95th percentile finds 35% more defects than the 5th percentile (38/28). Thus, the added benefit of using multiple individuals is the smaller variance between the "good" (95th percentile) and "bad" (5th percentile) scenarios.

Table 4. Number of average unique defects; 5th-95th percentiles in brackets

Testers	A TR	B TR	B NTR	2 *B TR	3 * B TR	4 * B TR	5 * B TR
1	5.85 (3-9)	7.58 (4-11)	11.28 (4-18)	12.07 (8-17)	15.11 (11-20)	17.44 (13-22)	19.32 (15-24)
2	10.07 (6-14)	12.07 (8-17)	16.98 (11-23)	17.44 (13-22)	20.93 (17-26)	23.59 (19-29)	25.77 (21-31)
3	13.36 (9-18)	15.11 (11-20)	20.55 (15-26)	20.93 (17-26)	24.73 (20-30)	27.61 (23-32)	29.95 (25-35)
4	16.04 (12-20)	17.44 (13-22)	23.24 (18-29)	23.59 (19-29)	27.61 (23-32)	30.64 (26-35)	33.07 (28-38)
5	18.21 (14-23)	19.32 (15-24)	25.41 (20-31)	25.77 (21-31)	29.95 (25-35)	33.07 (28-38)	35.58 (31-40)
6	20.1 (15-25)	20.93 (17-26)	27.24 (22-33)	27.61 (23-32)	31.93 (27-37)	35.11 (31-40)	37.62 (33-42)
7	21.75 (17-26)	22.32 (18-27)	28.83 (24-34)	29.21 (24-34)	33.62 (29-38)	36.84 (32-41)	39.37 (35-43)
8	23.22 (19-28)	23.59 (19-29)	30.21 (25-35)	30.64 (26-35)	35.11 (31-40)	38.35 (34-42)	40.87 (37-45)
9	24.57 (20-29)	24.73 (20-30)	31.43 (26-36)	31.93 (27-37)	36.44 (32-41)	39.69 (35-44)	42.18 (38-46)
10	25.78 (21-30)	25.77 (21-31)	32.52 (28-37)	33.07 (28-38)	37.62 (33-42)	40.87 (37-45)	43.36 (40-46)
...	...	...	...	...	...	...	...
40	43.06 (39-47)	40.87 (37-45)	44.39 (42-46)	-	-	-	-

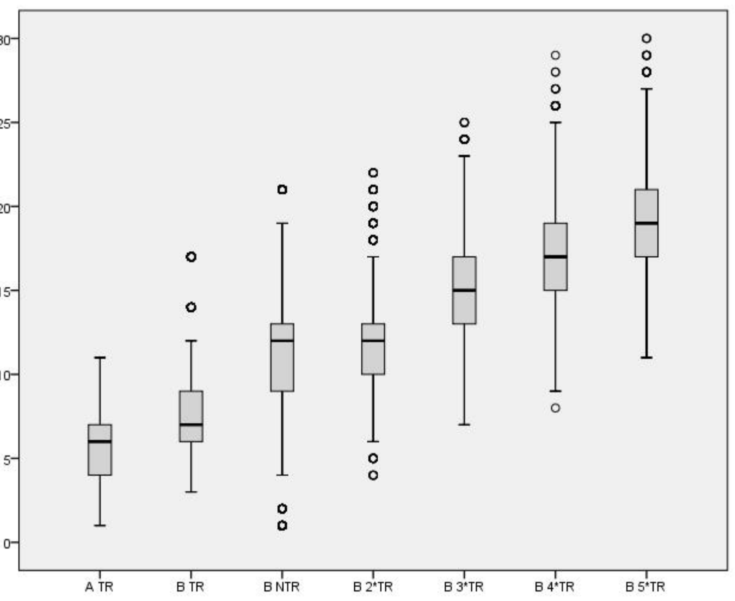


Figure 3. Boxplot of the number of defects found

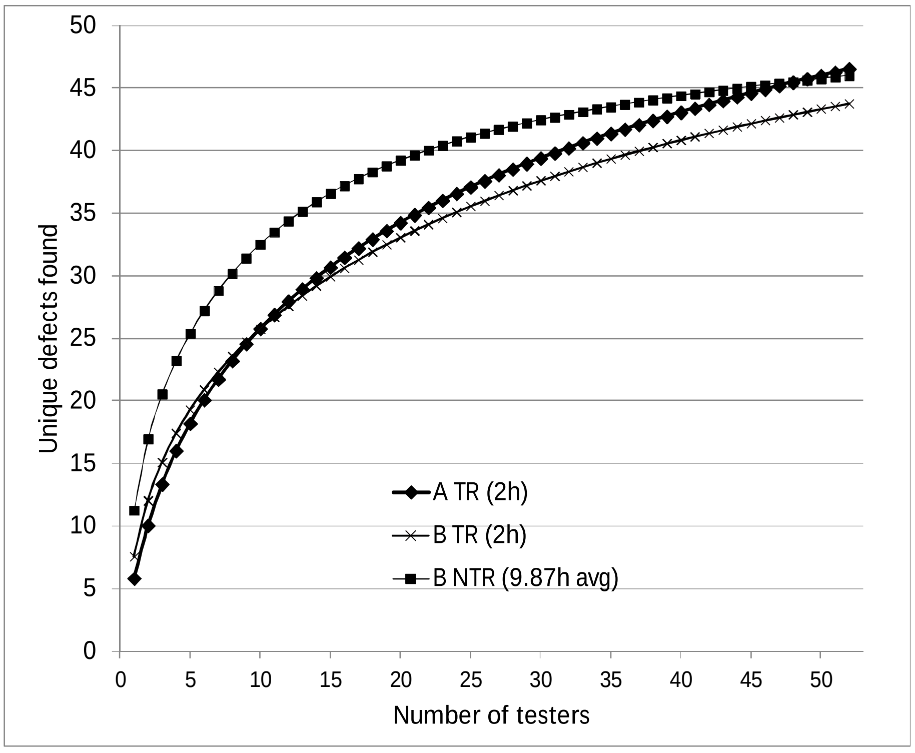


Figure 4. Number of unique cumulative defects found by testers

4.2. Validity and F-score

This section uses the validity and F-score, as defined in Section 3.5, to evaluate the performance of tester crowds under TR and NTR conditions. We use these two ways to address the duplicate defect reports described in Section 3.1. First, duplicate findings are not taken into account when calculating validity in Section 4.2.1. This simulates the case in which a software development organization has efficient mechanisms for duplicate handling and duplicates introduce no extra cost in the processing of defects after testing. Second, duplicates are considered as bad as invalid defect reports in Section 4.2.2. This simulates a case in which each duplicate defect has the same cost as invalid defect reports.

4.2.1. Duplicates as ignored

Table 5 and Figure 5 show how time-restriction improves validity when comparing single testers. Conditions A TR and B TR have similar validity numbers for a single tester, i.e., 82% for feature sets A and B. In condition B NTR, the validity of a single tester is lower, 73%. The difference between the validity in the B TR and the B NTR condition has an effect size (Cohen's *d*) of 0.55, indicating a medium effect.

In Figure 5, we can see that, as the number of testers is increased, effectiveness increases and validity decreases in all our conditions. The Table 5 is read the same way as Table 4. They both provide a comparison between the B NTR condition and multiple individuals in the B TR condition. In the table, we can see that, from the validity viewpoint, having 2-5 B TR testers (69.45-78.35%) is roughly equal to having a single NTR tester (73.30%), on average. The effect size range is from -0.24 to 0.31, indicating that there is a small effect and that the effect direction is dependent on the number of TR testers.

Table 5. Validity; 5th-95th percentiles in brackets (duplicates ignored)

Testers	A All	B TR	B NTR	2 * B TR	3 * B TR	4 * B TR	5 * B TR
1	82.17% (50%-100%)	82.18% (54%-100%)	73.30% (50%-100%)	78.35% (55%-95%)	74.83% (55%-93%)	71.98% (55%-88%)	69.45% (53%-85%)
2	80.18% (55%-100%)	78.35% (55%-95%)	67.00% (48%-87%)	71.98% (55%-88%)	67.16% (52%-82%)	63.15% (50%-77%)	59.84% (48%-73%)
3	78.16% (57%-100%)	74.83% (55%-93%)	61.52% (45%-79%)	67.16% (52%-82%)	61.41% (49%-75%)	56.96% (46%-69%)	53.36% (43%-64%)
4	76.38% (58%-94%)	71.98% (55%-88%)	57.07% (43%-73%)	63.15% (50%-77%)	56.96% (46%-69%)	52.35% (43%-63%)	48.57% (40%-58%)
5	74.62% (57%-91%)	69.45% (53%-85%)	53.46% (41%-69%)	59.84% (48%-73%)	53.36% (43%-64%)	48.57% (40%-58%)	44.76% (38%-53%)

6	72.95% (56%-90%)	67.16% (52%-82%)	50.46% (39%-64%)	56.96% (46%-69%)	50.4% (41%-60%)	45.46% (38%-54%)	41.63% (35%-48%)
7	71.32% (56%-87%)	65.02% (50%-80%)	47.84% (37%-61%)	54.48% (44%-66%)	47.75% (40%-57%)	42.81% (36%-50%)	38.98% (34%-45%)
8	69.89% (55%-85%)	63.15% (50%-77%)	45.57% (36%-58%)	52.35% (43%-63%)	45.46% (38%-54%)	40.51% (35%-47%)	36.68% (32%-42%)
9	68.57% (55%-83%)	61.41% (49%-75%)	43.52% (34%-55%)	50.4% (41%-60%)	43.44% (37%-51%)	38.49% (33%-44%)	34.69% (31%-39%)
10	67.32% (54%-81%)	59.84% (48%-73%)	41.67% (33%-52%)	48.57% (40%-58%)	41.63% (35%-48%)	36.68% (32%-42%)	32.91% (29%-37%)

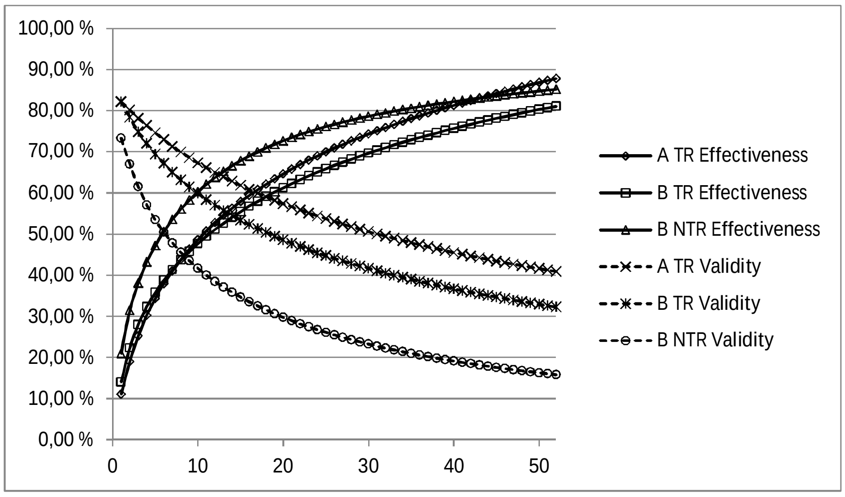


Figure 5. Effectiveness and validity and the number of testers (x-axis)

Figure 5 shows how effectiveness and validity behave as more testers are added, but it offers no advice regarding the sweet spot in terms of balancing effectiveness and validity. To combine these measures, a measure called the F-score was introduced in Section 3.5. Table 6 and Figure 6 show the results of the F-score analysis. In condition, A TR, the F-score peaks at 22 testers with a value of 60.71%, in condition B TR, the F-score peak is at 17 testers at 54.30%, and in condition B NTR, the peak is at 7 testers with 49.9%.

Time-restriction has a negative effect on the F-score at the individual level. B TR testers have a lower F-score of 23.74%, while B NTR testers have an F-score of 31.63%. The effect size of this difference is *d*=-0.90, indicating a large effect. When comparing 2-5 TR testers against a single B NTR tester, we find that TR testers produce higher F-scores (34.53%-46.97% vs. 31.63%). The effect sizes *d* are between small and very large 0.33-1.74. Thus, having time-restricted crowds produces a positive effect in terms of the F-score.

Table 6. F-score measures with peak values emphasized; 5th-95th percentiles in brackets (duplicates ignored)

Testers	A TR	B TR	B NTR	2 * B TR	3 * B TR	4 * B TR	5 * B TR
1	19.25% (10%-29%)	23.74% (13%-34%)	31.63% (13%-46%)	34.53% (25%-46%)	40.47% (31%-51%)	44.34% (35%-54%)	46.97% (38%-57%)
2	30.47% (19%-42%)	34.53% (25%-46%)	41.98% (32%-52%)	44.34% (35%-54%)	48.91% (40%-59%)	51.41% (42%-61%)	52.87% (44%-62%)
3	37.87% (27%-49%)	40.47% (31%-51%)	46.29% (38%-56%)	48.91% (40%-59%)	52.24% (43%-61%)	53.69% (45%-62%)	54.21% (46%-63%)
4	43.09% (32%-53%)	44.34% (35%-54%)	48.41% (40%-57%)	51.41% (42%-61%)	53.69% (45%-62%)	54.29% (46%-62%)	54.03% (46%-62%)
5	46.8% (37%-56%)	46.97% (38%-57%)	49.45% (42%-58%)	52.87% (44%-62%)	54.21% (46%-63%)	54.03% (46%-62%)	53.19% (46%-60%)
6	49.65% (40%-59%)	48.91% (40%-59%)	49.92% (43%-59%)	53.69% (45%-62%)	54.27% (47%-62%)	53.39% (46%-61%)	52.03% (46%-59%)
7	51.84% (42%-61%)	50.31% (41%-60%)	49.99% (43%-58%)	54.11% (46%-63%)	53.91% (47%-61%)	52.51% (46%-59%)	50.73% (45%-57%)
8	53.61% (44%-63%)	51.41% (42%-61%)	49.81% (43%-58%)	54.29% (46%-62%)	53.39% (46%-61%)	51.52% (45%-58%)	49.36% (44%-55%)
9	55.07% (46%-64%)	52.24% (43%-61%)	49.42% (42%-57%)	54.27% (47%-62%)	52.76% (46%-60%)	50.46% (45%-56%)	48.% (43%-53%)
10	56.24% (47%-65%)	52.87% (44%-62%)	48.92% (42%-57%)	54.03% (46%-62%)	52.03% (46%-59%)	49.36% (44%-55%)	46.66% (42%-51%)
...	...	...	...	...	...	...	...
17	60.14% (52%-68%)	54.30% (46%-62%)	44.22% (38%-51%)				
22	60.71% (53%-68%)	53.75% (46%-61%)	40.74% (35%-47%)	-	-	-	-

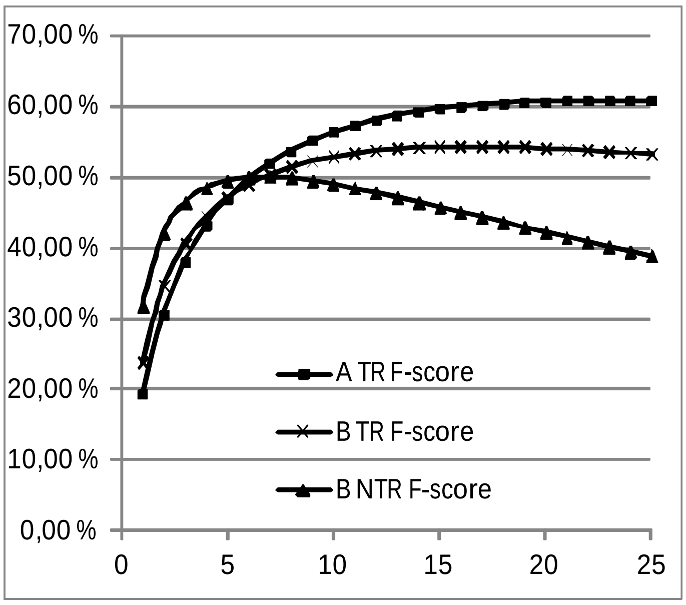


Figure 6. F-score measures; number of testers on the x-axis (duplicates ignored)

4.2.2. Duplicates as invalid defect reports

It is easy to understand that the number of duplicates quickly starts to dominate the share of defect reports as more testers are added to a crowd of testers. This is exemplified in Figure 7. In the previous section, duplicates were ignored by the validity and F-score analysis. In this section, duplicates are considered as bad as invalid defect reports.

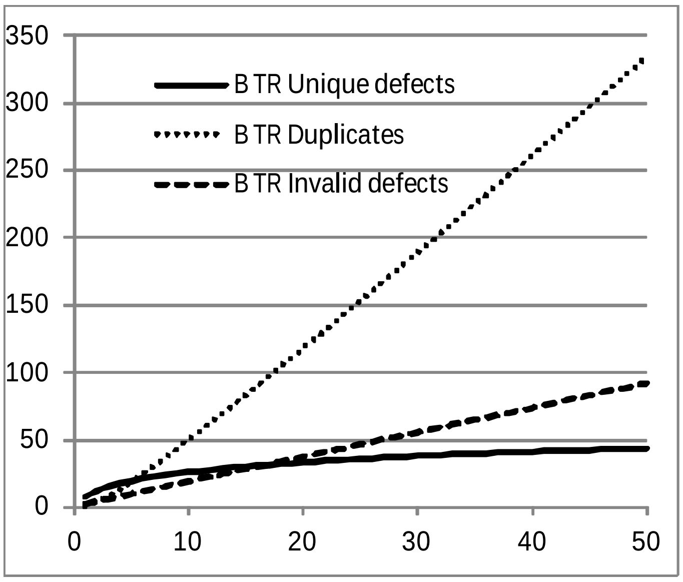


Figure 7. Number of defects (y-axis) and number of testers (x-axis) in condition B TR

Table 7 shows the same analysis as Table 5 but with duplicates produced by different individuals considered invalid defect reports. At the individual level, the effects of time pressure to validity are the same as in Section 4.2.1 because it requires multiple individuals to create duplicate defect reports.

When comparing a single B NTR tester against 2-5 B TR testers, we can see that condition B NTR has superior validity (73.3% vs. 65.22-41.69%). The effect sizes *d* of these differences are between -1.97 and -0.50, indicating a medium to a very large effect. Thus, when duplicates are considered invalid defect reports, it is better to use a single NTR tester rather than several TR testers from the viewpoint of pure validity.

Table 7. Validity averages; 5th-95th percentiles in brackets (duplicates as invalid)

Testers	A TR	B TR	B NTR	2 * B TR	3 * B TR	4 * B TR	5 * B TR
1	82.17% (50%-100%)	82.18% (54%-100%)	73.30% (50%-100%)	65.22% (47%-82%)	54.30% (41%-68%)	47.02% (37%-58%)	41.69% (33%-51%)
2	71.27% (50%-91%)	65.22% (47%-82%)	55.15% (40%-72%)	47.02% (37%-58%)	37.60% (30%-45%)	31.73% (26%-38%)	27.69% (23%-33%)
3	62.94% (47%-79%)	54.3% (41%-68%)	44.14% (33%-58%)	37.6% (30%-45%)	29.54% (24%-35%)	24.71% (21%-29%)	21.43% (18%-25%)
4	56.56% (44%-70%)	47.02% (37%-58%)	37.12% (29%-47%)	31.73% (26%-38%)	24.71% (21%-29%)	20.56% (17%-24%)	17.74% (15%-20%)
5	51.43% (41%-63%)	41.69% (33%-51%)	32.31% (26%-40%)	27.69% (23%-33%)	21.43% (18%-25%)	17.74% (15%-20%)	15.26% (13%-17%)
6	47.28% (38%-57%)	37.60% (30%-45%)	28.80% (23%-36%)	24.71% (21%-29%)	19.04% (16%-22%)	15.69% (14%-18%)	13.45% (12%-15%)

7	43.82% (36%-53%)	34.34% (28%-41%)	26.06% (21%-32%)	22.40% (19%-26%)	17.17% (15%-19%)	14.11% (13%-16%)	12.06% (11%-13%)
8	40.93% (34%-49%)	31.73% (26%-38%)	23.87% (20%-29%)	20.56% (17%-24%)	15.69% (14%-18%)	12.85% (11%-14%)	10.96% (10%-12%)
9	38.47% (32%-46%)	29.54% (24%-35%)	22.04% (18%-27%)	19.04% (16%-22%)	14.48% (13%-16%)	11.82% (11%-13%)	10.06% (9%-11%)
10	36.35% (30%-43%)	27.69% (23%-33%)	20.49% (17%-24%)	17.74% (15%-20%)	13.45% (12%-15%)	10.96% (10%-12%)	9.30% (9%-10%)

Table 8 and Figure 8 show the results of the F-score analysis when duplicate defect reports are considered invalid. In condition A TR, the F-score peaks at 7 testers with a value of 42.22%, in condition B TR, the F-score peaks at 5 testers at 38.36%, and in condition B NTR, the peak is at 3 testers at 40.26%. A comparison to the previous section where duplicates are not considered invalid shows that the peaks in the F-scores are lower and come with a lower number of testers (3-7 vs. 7-22).

When duplicates are seen as invalid defects, there is no clear difference between using TR or NTR testers. In Table 8, we see that a single B NTR tester offers an inferior F-score in comparison to 2-5 B TR testers (31.63% vs. 33.04%-38.36%). The effect sizes *d* between the differences are from low to medium: 0.16-0.76. However, when we compare two B NTR testers against 4-10 B TR testers, the results are reversed, as we see that the NTR testers are superior (39.22% vs. 34.99%-38.12%). The effect size *d* of these differences ranges from low to medium: -0.20 to -0.77. Thus, the results between TR and NTR are mixed in this case.

Table 8. F-score measures with peak values emphasized; 5th-95th percentiles in brackets (duplicates as invalid)

Testers	A TR	B TR	B NTR	2 * B TR	3 * B TR	4 * B TR	5 * B TR
1	19.25% (10%-29%)	23.74% (13%-34%)	31.63% (13%-46%)	33.04% (24%-44%)	36.71% (28%-46%)	38.12% (30%-46%)	38.36% (31%-46%)
2	29.73% (19%-41%)	33.04% (24%-44%)	39.22% (30%-48%)	38.12% (30%-46%)	38.05% (31%-45%)	36.67% (30%-43%)	34.99% (29%-41%)
3	35.74% (25%-46%)	36.71% (28%-46%)	40.26% (33%-48%)	38.05% (31%-45%)	35.84% (30%-42%)	33.28% (28%-39%)	30.88% (26%-35%)
4	39.19% (30%-48%)	38.12% (30%-46%)	39.42% (33%-47%)	36.67% (30%-43%)	33.28% (28%-39%)	30.16% (26%-34%)	27.5% (24%-31%)
5	40.99% (33%-49%)	38.36% (31%-46%)	37.98% (32%-45%)	34.99% (29%-41%)	30.88% (26%-35%)	27.5% (24%-31%)	24.77% (22%-28%)
6	41.91% (34%-50%)	38.05% (31%-45%)	36.41% (31%-43%)	33.28% (28%-39%)	28.79% (25%-33%)	25.27% (22%-28%)	22.54% (20%-25%)
7	42.22%	37.42%	34.84%	31.65%	26.91%	23.38%	20.7% (19%-

	(35%-50%)	(31%-44%)	(30%-41%)	(27%-36%)	(23%-30%)	(21%-26%)	23%)
8	42.19% (35%-50%)	36.67% (30%-43%)	33.31% (29%-39%)	30.16% (26%-34%)	25.27% (22%-28%)	21.76% (19%-24%)	19.14% (17%-21%)
9	41.93% (35%-49%)	35.84% (30%-42%)	31.86% (27%-37%)	28.79% (25%-33%)	23.83% (21%-27%)	20.37% (18%-22%)	17.82% (16%-19%)
10	41.51% (35%-48%)	34.99% (29%-41%)	30.49% (26%-35%)	27.5% (24%-31%)	22.54% (20%-25%)	19.14% (17%-21%)	16.67% (15%-18%)

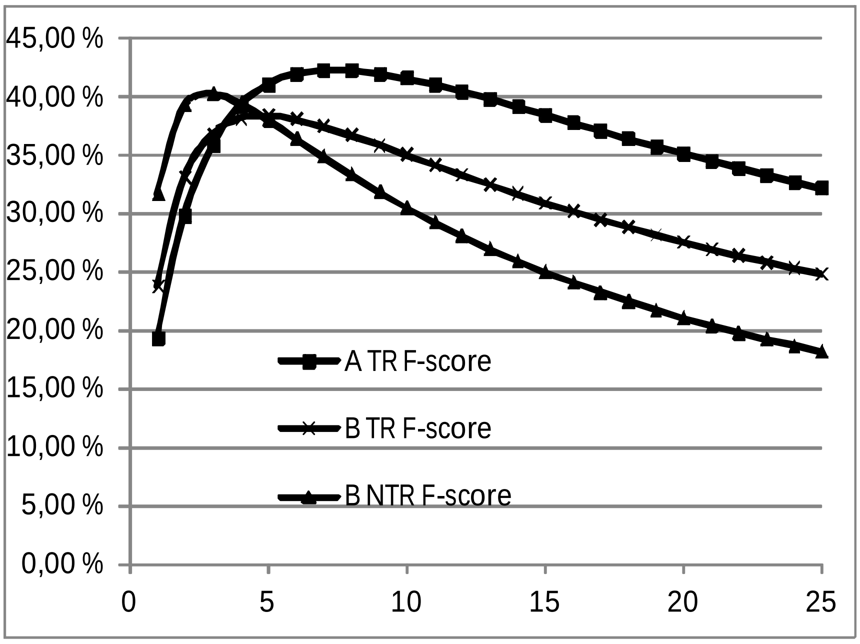


Figure 8. F-score measures; number of testers on the x-axis (duplicates as invalid)

5. Discussion

5.1. RQ1: The effect of adding individual testers to a crowd of testers

In this section, we answer the first research question – "What is the effect on the defect detection performance of adding individual testers to a non-communicating crowd of testers?" – from two viewpoints. First, Section 5.1 answers it from the viewpoint of unique defects and effectiveness. Section 5.2 answers it from the viewpoints of validity and F-score.

5.1.1. Viewpoint 1 – Number of unique defects and effectiveness

Regarding RQ1 from the viewpoint of unique defects, we can state that the increase in the number of individuals increases the defect detection capability of a crowd. The relationship between the increase in the number of individuals and number of unique defects found, on average, is asymptotic with respect to the total number of defects. Thus, based on this study, manual software testing at the system level is, in terms of Steiner's taxonomy [12], an *additive task with a ceiling effect* caused by the exhaustion of defects. Analysis of prior works (see Section 2.2) shows that other software verification and validation tasks, in addition to manual software testing, including at least software reviews and usability inspections, are also additive tasks with ceiling effects. We think that uniting different software verification and validation tasks under the taxonomy of Steiner, as we have done in this paper, helps in understanding the characteristics and commonalities of these tasks. The taxonomy helps in comparing the tasks with other software engineering tasks and provides a theoretical basis to conduct better studies.

The reason for the additive nature of these tasks comes from the fact that different testers or reviewers find different defects [6, 7]. Furthermore, we hypothesize that this difference between individuals is caused by two fundamental factors of software testing: *coverage* and *oracles*. First, the coverage of verification and validation tasks increases as more individuals are added, as each individual will introduce variations in the way the task is performed, and due to these differences in test design and random variations, different individuals detect different sets of defects. The second reason for the additive nature of verification and validation tasks is the *defect detection capabilities (= oracle)*, which vary between humans. These capabilities originate from psychological factors, e.g., cognitive styles [22], and knowledge levels of individuals – e.g., various types of personal knowledge contributes the detection of defects in testing [28] – and additionally, common sense would suggest that different viewpoints (developers, testers, users) will lead to the detection of different defects. These factors originating from human cognition were presented in Section 2.2.1.

Unfortunately, our data gives no way to segregate the effect of these factors as we cannot link the defect detection effectiveness to a) psychological factors explaining the effectiveness of human oracles, or to b) the coverage increase caused by the individual variation in the manual test execution. Past work has shown that the quality of a test oracle affects the defect detection effectiveness [60]. We also perceived that, in our experiment, even difficult to detect defects on functionalities that were clearly described in the user manual were often detected.

Finally, we discuss the question of high practical relevance, i.e., how much one gains when going from one individual to two individuals in software verification and validation. Often, resources are scarce, and people working in the industry must be able to justify actions such as using a second individual for a task that might have previously been performed by only a single individual. On the other hand, in the field of medicine, there is a practice called "second opinion" that employs more than one physician to reduce the possibility of an incorrect diagnosis [19]. In

Table 9, we have summarized the benefits of having a second individual in software verification and validation based on this study and prior works. It presents combined data from several data sources and, thus, helps in understanding the additive nature of verification and validation tasks over different verification and validation methods. For the table, we have included data only from sources where a number of individuals is 10 at a minimum (this allows 45 combinations for the first two individuals) to avoid biases from samples that are too small. We can see that, on average, adding a second individual to do the same V&V task results in a roughly 50% increase in the effectiveness of finding unique defects. The variation between the sources is rather large, as the benefit of the second individual varies between 25% and 78%. The results of the studies are consistent in that there is a positive effect on the defect detection effectiveness when a second individual is added to a V&V task. However, the decision regarding whether the expected gains of 25% to 78% more unique findings are enough to cover the additional cost remains a context-specific decision.

Table 9. Benefit of adding a second individual to verification and validation tasks

Data source (number of individuals)	**V&V task type**	**Effectiveness % of defects found by 1 individual**	**Effectiveness % of defects found by 2 individuals**	**Increase in the % of defects found (1->2)**
A TR (79)	Manual, system-level, black box testing with 2h time restriction (feature set A)	11.02%	19.00%	72.43%
B TR (79)	Manual, system level, black box testing with 2h time restriction (feature set B)	13.95%	22.18%	58.99%
B NTR (51)	Manual, system level, black box testing with no time restriction (feature set B)	20.93%	31.35%	49.79%
WRBM S (47) [7]	Automating, unit level, white box test cases, with structural coverage criteria (max. time 3h; could leave earlier if done)	55.00%	71.80%	30.55%
WRBM F (47) [7]	Automating, unit level, white box test cases, with functional coverage criteria (max. time 3h; could leave earlier if done)	57.90%	72.30%	24.87%
BH 2h (180) [2]	Reading requirement document; defects found after 2 hours	9.00%	16.00%	77.78%
BH 4h (180) [2]	Reading requirement documents; defects found after 4 hours	18.00%	29.50%	63.89%
BH 6h (180) [2]	Reading requirement documents; defects found after 6 hours	21.00%	34.50%	64.29%
Ni N (31) [27]	Heuristic evaluation of the usability of a banking system with novice evaluators	22%	33.50%	52.27%

Ni R (19) [27]	Heuristic evaluation of usability of banking system with usability expert evaluators	40.50%	62.00%	53.09%
Ni D (14) [27]	Heuristic evaluation of the usability of a banking system with usability and domain expert evaluators	59.50%	81.00%	36.13%
FAS W (15) [33]	Usability inspection session lasting 2h with work-domain experts	25.00%	37.50%	50.00%
FAS U (12) [33]	Usability inspection session lasting 2h with usability expert evaluators	40.50%	55.00%	35.80%
Averages		**30.33%**	**43.51%**	**51.53%**

5.1.2. Viewpoint 2 – Validity and F-score

Although adding more testers brings the positive effect of finding more defects, it also amplifies a negative effect, as the relative share of invalid defect reports and duplicate defect reports increases, as depicted in Figure 7. For example, the 10th tester will find, on average, a small share of unique defects (1.0-1.2), a considerable share of invalid defects (1.4-4.4), and a big share of duplicates (4.6-10.1). In Section 3.5, we introduced effectiveness, *validity*, and the *F-score* as measures for studying testers' performance. The effectiveness of a crowd increases when the size increases, while validity decreases when size increases; thus, neither of these measures is particularly salient when trying to determine the optimal crowd size.

The F-score is a measure that provides balance between effectiveness and validity and the peak values of the F-score tell us about the sweet spots regarding the number of testers. We showed that the peaks of F-score are affected by the way in which duplicates are handled (see Figure 6 and Figure 8). In our case, the F-score produces two values of interest: first, the peak value of the F-score itself and, second, the number of testers where the peak value appears. For example, if duplicates are ignored (i.e., the case where a company has efficient duplicate handling mechanisms), then the peak values of the F-score are between 50-61%, and they are found with 7-21 testers in our conditions, and when duplicates are invalid defects (i.e., the case when a company has poor duplicate handling mechanisms), then the peak values of the F-score are between 38-42%, and they are found with 3-7 testers in our conditions.

An important theme is how we can influence the F-score peaks. Effective duplicate handling tools, e.g. latent semantic analysis to identify duplicates [61], reduces duplicates and, thus, increases the F-score value and allows organizations to benefit from larger crowds. Instructions to individual testers can also affect the number of duplicates; e.g., before submitting a defect report, each tester is required to perform an extensive search to ensure that such a defect has not been previously reported. Reducing the number of invalid defect reports will also improve the F-score and affect the number of testers used. This reduction in invalid defect reports can, for example, be achieved by instructing each tester to double-check their defects' reproducibility and the defect report itself. Another way to reduce the number of invalid defect reports is to allow only automated defect reporting, which has the additional benefit of allowing highly effective automatic processing of defect information that can then be used in statistical quality assurance, e.g., in [62]. Prior work has shown that beta testing [9] is an effective verification and validation measure with a high number of beta-testers (>1,000) [10]. However, we see that such a scenario requires either very strict instruction on what to report and how to report defects or robust automation. Otherwise, the processing of duplicates and invalid defects becomes a huge source of overhead, as shown in Figure 7. To summarize, this discussion highlights that the size of the crowd depends on how many duplicates and invalid defects are produced by the crowd and how are they handled. Thus, even if one can add testers virtually for free, e.g., by inviting more customers to test a highly popular product that already has a customer test program ready, it might not be economically wise to maximize the number of testers, as it will maximize the invalid and duplicate defect reports as well.

Finally, we showed that in none of the studied conditions was a maximal F-score achieved by using a single tester. The smallest number of testers to achieve the peak F-score value was three, as shown in Tables 6 and 8. Thus, it is advisable to have more than a single tester to perform the testing of a particular feature. Furthermore, we think that future studies of software verification and validation should report also the validity and F-score to provide a more complete picture of the benefits and drawbacks of each verification and validation technique. Typically, in studies of software testing and software reviews, only effectiveness, i.e., the percentage of unique valid defects found, is reported.

5.2. RQ2: The effect of time restriction between TR and NTR testers

In this section, we answer the second research question: "What is the effect of time restriction on the defect detection performance between TR and NTR testers?" This question is covered on two levels: on the individual level and on the crowd level when controlling for effort. Similarly, with the first research question, the results regarding the second research question are next discussed form two viewpoints: first from the viewpoint of the number of unique defects and, second, from the viewpoint of the validity and F-score measures.

5.2.1. Viewpoint 1 – Number of unique defects

On individual level, NTR testers find more defects than TR testers (11.28 vs. 7.53), and this difference has an effect size, a Cohen's *d* of -0.96, indicating a large effect. This is not surprising as, on average, the NTR testers spent 9.87 hours testing, while the TR testers were restricted to 2 hours. Considering efficiency, we found that TR testers had clearly higher efficiency (1.33 vs. 3.79 defects/h). One could assume that this is caused by having a set of defects that are much easier to detect than other defects, and the TR testers would prominently detect these easily detectable defects while only rarely detecting other defects. This would lead to higher efficiency but lower effectiveness. If this is the case, then the effort-controlled crowd comparison should produce no differences between TR and NTR testers.

On the crowd level, time-restricted (TR) testers outperform non-time-restricted (NTR) testers when controlling for effort. Five TR testers equaled the effort of 10 hours of testing and resulted in 19.32 found defects, while a single NTR tester used 9.87 hours and found 11.28 defects, on average. This results in a very large effect size between crowds, a Cohen's *d* of 2.04. Therefore, it seems that the TR testers' efficiency was not caused by detecting only easily detectable defects. However, it could be that what is easily detectable varies between individuals; thus, the TR testers' efficiency could still be caused by detecting only easily detectable defects, but the large variation in "what is easily detectable" between individuals would explain the TR crowds' effectiveness. Nevertheless, based on our data, it would be beneficial to have several testers who are under time pressure because we can either find roughly the same amount of defects with 59% less effort, or we can use the same effort to find 71% more defects.

We could not find any prior work directly related to these results in the software engineering domain. Biffle and Halling [2] studied software inspections and the defects found by people who read for 2, 4, 6, and 8 hours. In that study, there were no major differences in performance whether there was a single inspector or multiple inspectors with the same amount of time (see Section 2.2.2). However, the difference from our study is that they did not impose time restrictions, as each individual could spend as much time reading as desired. The individual time groups in that study were formed from intermediate results after reading for the first 2, 4, 6, and 8 hours. Thus, we think that they cannot be compared with ours, as the people in our time-restricted group had a real deadline.

The effect of time pressure has been studied in the software project context (see Section 2.3), and it has been found that some time pressure, studied as budget and schedule pressure separately, is beneficial and can result in savings [4]. This project-level data does not allow for detailed comparison with our results, but both the findings of this study and those of [4] support the hypothesis that time pressure is beneficial.

The only small task time pressure study in the software engineering context is by Topi et al. [46], who found that time pressure did not have an effect on performance. We believe that the difference from our results is due to differences in tasks; Topi et al. studied a database query creation task, i.e., a problem-solving task that, according to Steiner’s taxonomy [12], would be disjunctive and comparable to programming (see Table 1). We, on the other hand, studied a testing task, i.e., a searching task that is classified as additive (see Table 1).

This contradiction of Topi et al. [46] suggests that time pressure would be useful only in additive tasks where it is possible to go at different speeds. In other words, it would not be possible to increase speed in a problem-solving task, as one cannot solve the problem any faster, whereas it would be possible to increase speed in a searching task. However, if one chooses to increase speed, this would lead to decreased effectiveness on an individual level but also to increased efficiency, as shown by our results. This result of increasing efficiency and decreasing effectiveness under time pressure was previously found in an experiment of the tests of details of inventory that come from the domain of accounting [47]. According to an accounting textbook [63], test of details of inventory contains tasks including *trace counts noted at inventory count to final listing*, and *for selected items, inspect sales subsequent to year end to ensure prices exceed inventory costs*. Based on the verbs used in the task description (trace, inspect, ensure) it seems that accounting tasks could be similar to software verification and validation tasks. Thus, there is good reason to believe that the effects of time pressure would also be similar in these tasks, which strengthens the finding that time pressure increases efficiency but decreases effectiveness in software verification and validation.

Time pressure is one of the possible factors explaining why using multiple TR testers leads to higher effectiveness than using a single NTR tester when the effort is equal between the crowds. Next, we list the three additional hypotheses to explain the difference. See Table 10 in Section 6 for a summary of all the hypotheses and implications for practitioners. First, as discussed in Section 5.1.1, different testers find different defects. Second, we believe that having more “fresh eyes” helped the TR testers to perform better. With “fresh eyes,” we are refereeing to phenomena when individuals become blind to defects in the areas they have been working in for a long time; e.g., after some hours of testing or debugging, individuals are less likely to detect new defects in the same features of the software. In psychology, this phenomenon is known as habituation [64]. In general terms, habituation means that, if stimuli persist in the environment, then, over time, individuals will no longer pay attention to it. An interesting way to counter habituation in verification and validation is to make individuals think of the reasons behind each defect and then have them use this information to find new defects, as shown in [65]. It was also shown in [65] that simply telling the individuals that more defects exist and giving them more time for searching results in a very small increase in the number of defects found. In other words, when a certain individual cannot find more defects, one can either bring in a second individual or give the first individual a mental tool that would alter their defect searching and detection process. Third, we think it is highly likely that the NTR testers suffered from overspending of effort since they did not know when to stop testing and, to err on the side of caution, they would have been more likely to do too much testing than too little. This would show up in high effectiveness but poor efficiency. The question of "when to stop testing" remains a mystery to the software testing community, although many heuristics have been provided, e.g., in [66].

5.2.2. Viewpoint 2 – Validity and F-score

On the individual level, TR testers have higher validity (d=0.55) and but lower F-scores than NTR testers (d=-0.90). This means that, although the ratio of valid defects is better for a single TR tester, the overall performance of NTR testers is better. It is clear that time restriction has a negative effect on performance at the individual level.

At the crowd level, the results are mixed. When duplicates were ignored, NTR testers had better overall performance in terms of F-scores (see Table 6), but there was no difference in validity between a crowd of TR testers and a single NTR tester. When we consider duplicate defect reports as invalid defect reports, the TR and NTR testers offer similar overall performance in terms of F-scores (see Table 8), but the validity was better for NTR testers. Thus, when an effective duplicate handling policy and tool are in place, crowds of TR testers deliver better overall performance, and when no such tools are in place, the performance between TR and NTR testers is equal. This finding is an example case of the discussion in Section 5.1.2, where we claimed that the optimal size of the crowd is determined by how many invalid and duplicate defects the crowd produces and how are they handled.

5.3. Threats to validity

Next, we discuss the internal, external, construct, and conclusion validity. We have used the list of validity threats by Wohlin et al. [67] as a checklist, and in this section, only the threats that are present in this experiment are discussed. Additionally, we point out the research question that each validity threat is linked to, and this analysis shows that more of our validity threats are linked to RQ2 than to RQ1.

Internal validity is concerned about causal relationships between the treatment and the variable measured as an outcome. None of the single group threats listed in [67] is present since the experimental procedures were carried out in two successive years with different software engineering students participating. However, there is a multi-group threat that deals with a selection of individuals, and it affects RQ2. Our participants came from successive years, and we studied the background variables of the participants, number of credits, work experience, etc., to make sure that they would be samples from the same population (see Table 2). Nevertheless, there is still a possibility that the groups are unequal in some latent respect. Social threats to validity are present when groups are somehow aware of each other. For example, a control group may be demoralized by using the less desirable treatments. In our case, we used students from successive years; thus, it was not possible for the participants in the first year to be aware of what would happen in the next year. On the other hand, the second-year participants were not made aware of the actions taken in the previous year.

Construct validity concerns whether the experimental design and measures actually measure what the researchers aim to measure. We used the number of reported defects as our measure of the results of testing, and it affected RQ1 and RQ2. This is one limited viewpoint regarding the results and benefits of testing activities that does not account for, e.g., the differences in the types of reported defects. The defect counts are, however, commonly used metrics in software engineering research and allow us to compare our results to other research. We experienced mono-operation bias as, in both years, the students tested the same features of the jEdit text editor, and it affected both RQ1 and RQ2. To overcome this, different types of software should be used for testing. For example, had we used software that was less familiar to the students, it is likely that the performance of TR testers would have declined due to their spending more time learning the software rather than executing the tests. We also experienced confounding constructs from using student subjects, and it affected both RQ1 and RQ2. For example, had we used experienced testers, it is more likely that they would not have spent so much time in the NTR condition, as such individuals would possess more knowledge on when to stop testing. Evaluation apprehension means that it is human nature to look better when evaluated. This could affect how the usage of time was reported in the second year, and it affected RQ2. In the second year, we collected the effort used in testing based on what the students reported in comparison to actually forcing them to work under supervision. The students were aware that no points were given based on how fast or slow they were; thus, they had no direct reason to lie. Still, it is possible that time reporting is biased, but unfortunately, we do not know whether it is biased toward larger or smaller values. For example, some students might have wanted to look good in the eyes of the course staff by reporting too few hours – "See, I completed this task very fast" – or by reporting too many hours – "See, I have worked very hard on this."

Regarding conclusion validity, we experienced issues regarding the reliability of the treatment, and it affected RQ2. Often in experimental design, one wishes to keep experimental treatments as close to each other as possible. We were interested in comparing how a crowd of time-restricted testers fared against testers without time restrictions. Thus, the conditions became more different from a typical experimental setup. In the first year, the testing was performed in computer rooms under the supervision of the course staff. In the second year, the participants were allowed to work on their own. One could argue that, in the second year, we should have organized supervised testing without time restrictions. However, since the average time used was 9.87h by 51 students, this means that we would have needed to book several computer classrooms over several days and allowed the students to continue working as long as they liked. Even though there are practical difficulties in such arrangement, it should be considered in the future.

Finally, external validity concerns how our results generalize to industrial practice, and it affected RQ1 and RQ2. First, we used students, who are comparable to junior testers in the software industry. However, the additive nature of verification and validation tasks when performed by industrial experts can be seen in studies of software usability [27, 33].

5.4. Further work

Based on this study, there are several avenues for further work. First, few software engineering experiments have assessed time pressure in relation to software engineering tasks. With small modifications, i.e. limiting the amount of time available in the experiment, many existing experimental designs could be used to study the effect of time pressure in different software engineering tasks, such as reviews, software maintenance tasks, and pair-programming. Furthermore, it is imperative to study the different levels of time pressure: small, medium, and high. In this study, we had only two conditions: time pressure and no-time pressure. Second, an interesting research question for future research is to study what kind of differences coordinated crowds would have in testing performance in comparison to non-communicating crowds. Although common sense would suggest that coordinated crowds would do better, a counter-argument can be formulated based on Surowiecki [8], who claimed that the performance of a crowd drops with an increase in communication between individuals; i.e., individuals lose their independence and start to bias their opinions based on the views of the others, so, overall, the wisdom of the crowds is reduced to the wisdom of a few or a consensus among mediocre members. Third, crowd-based testing should also be studied with industrial case studies, for example, in a bug bash [68], which is a testing activity performed by group of people. Fourth, prior works have found that different individuals find different defects, but an explanation for this phenomenon is still mostly lacking. We think that these differences can be explained by studies focusing on human cognition, knowledge, and organizational roles. Knowledge of these factors would help the industry in selecting the most suitable individuals for verification and validation tasks. For example, what kind of psychological tests can explain performance in verification and validation tasks? On the other hand, what kind of knowledge is needed for detecting defects; is domain knowledge more valuable than general testing knowledge, or can a crowd of novice testers outperform a single expert tester? What is the effect of habituation [64], i.e., the "fresh eyes" effect [68], on defect detection performance? Furthermore, past work on software testing has shown that increases in test suite size or diversity lead to increased defect detection [69]. The results of this paper support the former results regarding the effect of diversity in testing by showing that an increase in the number of testers leads to increased defect detection effectiveness. Future studies should look to determine whether cognitive diversity leads to increased test suite coverage or defect detection effectiveness. Fifth, the question of when to stop testing has been studied previously, and software testing textbooks often suggest heuristics for it. However, studies of its application at the level of individual testers have, to our knowledge, not been conducted.

6. Conclusions

This paper makes four contributions. First, we present evidence that manual software testing is an additive task, meaning that adding more individual testers increases the number of unique defects found. The number of unique defects found increases when more testers are added, and it asymptotically approaches the total number of existing defects. Traditional thinking about software testing suggests that a feature should be tested only once at each testing level, as it focuses on maximizing efficiency and coverage. Our results suggest that it is better to have several testers perform redundant testing of the same feature, as different individuals are likely to detect different defects. We connect the finding of the additive nature of software testing to the general theoretical background of different task types [12] and to two other fields of software verification and validation: software reviews and usability inspections. We show that a similar additive pattern with a ceiling effect exists in software testing, software reviews, and usability inspections. In general, it seems that software testing, usability inspections, and software reviews are tasks with similar properties; thus, there is plenty that researchers working in these fields can learn from each other. For example, in the area of usability inspection, it has been found that people with a particular cognitive trait find significantly more usability problems than people without this trait [22]. It would seem natural that such a trait would have a significant impact in the area of manual software testing and software reviews.

Second, this paper presents a novel attempt to study empirically how effort should be divided between individuals in software testing; i.e., is it better to have one individual do a thorough job or have multiple individuals work under time pressure when controlling for total effort? The answer, based on our data, is that multiple time-pressured individuals deliver superior effectiveness (71% higher) with equal total effort. The hypotheses for this result and its implications for practitioners are shown in Table 10. Although our results seem very promising for time-pressured crowds, we need to be cautious, as more studies are needed, and the limitations of this study need to be addressed in future works.

Third, our experiment shows that time pressure in software testing increases efficiency at the individual level, but it also decreases effectiveness at the individual level. To the authors' best knowledge, this is the first experiment on the effect of time pressure in the software verification and validation context (time pressure studies in other areas were reviewed in Section 2.3). Further studies of time pressure in this context are needed, as time pressure is a highly relevant factor in the software industry.

Fourth, based on the F-score analysis, we show and discuss how the optimal number of testers is affected by the number of duplicate and invalid reports produced by the testers and by the effectiveness of the tools and policies to prevent and handle the invalids and duplicates. Effective preventing and handling mechanisms make it possible to employ larger crowds of testers and achieve superior results. In this analysis, we introduce commonly used measures in the information retrieval domain — *effectiveness* (the percentage of all/known defects found), *validity* (the percentage of real defects among all findings), and the *F-score* (harmonic mean of effectiveness and validity) — and show how they can be used to measure software testing. It is suggested that future studies should also present the validity and F-score measures, which are often ignored in prior works (see Section 2.2), to provide a more transparent view of the results and allow easier comparison between studies.

Table 10. Hypotheses and implications for practitioners for why a crowd of TR testers outperformed NTR testers.

Hypothesis	Explanation	Implications for Practitioners
Individuals find different defects	Different individuals are likely to detect different defects due to individual difference stemming from	Use a heterogeneous crowd of individuals in defect detection tasks. Try to find defects from different viewpoints.

	psychology, knowledge, and viewpoints.	
Time pressure	Time pressure leads to a more efficient way of working. This efficiency is caused by working faster and by focusing only on the task at hand.	Use time pressure, but not constantly, as it may backfire in the form of a high staff turnover rate. One can also use time pressure to have fun: "Let's compete to see who finds the most new defects in one hour."
Habituation	One becomes blind to defects in areas one has been working in for a long time.	Find a colleague who can review and test your area, while you return the favor. Let the area you are working in rest and re-visit it later.
Overspending	"When to stop testing" is a mystery; thus, it easy to continue testing with an unproductive test strategy that does not reveal defects.	Try to recognize when testing is not producing new information with the chosen strategy, and stop testing or change the testing approach, viewpoint, or the tester.

7. Acknowledgements

This work was supported by ELLIIT, the Strategic Area for ICT research, funded by the Swedish government, and by TEKES, the Finnish Funding Agency for Technology and Innovation.

8. References

[1] E. Raymond, "The cathedral and the bazaar," *Knowledge, Technology & Policy,* vol. 12, no. 3, 1999, pp. 23-49.

[2] S. Biffl and M. Halling, "Investigating the defect detection effectiveness and cost benefit of nominal inspection teams," *Software Engineering, IEEE Transactions on,* vol. 29, no. 5, 2003, pp. 385-397.

[3] M.V. Mäntylä, K. Petersen and D. Pfahl, "How many individuals to use in a QA task with fixed total effort?" *International Symposium on Empirical Software Engineering and Measurement,* 2012, pp. 311-314.

[4] N. Nan and D.E. Harter, "Impact of budget and schedule pressure on software development cycle time and effort," *Software Engineering, IEEE Transactions on,* vol. 35, no. 5, 2009, pp. 624-637.

[5] F. Elberzhager, A. Rosbach, J. Münch and R. Eschbach, "Reducing test effort: A systematic mapping study on existing approaches," *Information and Software Technology,* vol. 54, no. 10, 2012, pp. 1092-1106.

[6] N. Juristo, A.M. Moreno and S. Vegas, "Reviewing 25 years of testing technique experiments," *Empirical Software Engineering,* vol. 9, no. 1, 2004, pp. 7-44.

[7] M. Wood, M. Roper, A. Brooks and J. Miller, "Comparing and combining software defect detection techniques: a replicated empirical study," *ESEC '97/FSE-5: Proceedings of the 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on foundations of software engineering,* 1997, pp. 262-277.

[8] J. Surowiecki, *The wisdom of crowds: Why the many are smarter than the few and how collective wisdom shapes business, economies, societies, and nations,* Doubleday Books, 2004.

[9] R.J. Dolan and J.M. Matthews, "Maximizing the utility of customer product testing: beta test design and management," *J. Prod. Innovation Manage.,* vol. 10, no. 4, 1993, pp. 318-330.

[10] C. Jones, "Software defect-removal efficiency," *Computer,* vol. 29, no. 4, 1996, pp. 94-95.

[11] M.V. Mäntylä, J. Itkonen and J. Iivonen, "Who Tested My Software? Testing as an Organizationally Cross-Cutting Activity," *Software Quality Journal,* vol. 20, no. 1, 2012, pp. 145-172.

[12] I.D. Steiner, *Group Process and Productivity,* New York, New York, USA: Academic Press, 1972.

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

[14] I. Jacobson and I. Spence, "Why We Need a Theory for Software Engineering," *Dr. Dobb's,* October 02 2009. 2009,

[15] J.E. Hannay, D.I.K. Sjoberg and T. Dyba, "A systematic review of theory use in software engineering experiments," *Software Engineering, IEEE Transactions on,* vol. 33, no. 2, 2007, pp. 87-107.

[16] V. Balijepally, R. Mahapatra, S. Nerur and K.H. Price, "Are two heads better than one for software development? The productivity paradox of pair programming," *MIS Quarterly: Management Information Systems,* vol. 33, no. 1, 2009, pp. 91-118.

[17] P. Croskerry, "The importance of cognitive errors in diagnosis and strategies to minimize them," *Academic Medicine,* vol. 78, no. 8, 2003, pp. 775.

[18] M. Graber, R. Gordon and N. Franklin, "Reducing diagnostic errors in medicine: what's the goal?" *Academic Medicine,* vol. 77, no. 10, 2002, pp. 981.

[19] J.D. Kronz, W.H. Westra and J.I. Epstein, "Mandatory second opinion surgical pathology at a large referral hospital," *Cancer,* vol. 86, no. 11, 1999, pp. 2426-2435.

[20] B.E. Teasley, L.M. Leventhal, C.R. Mynatt and D.S. Rohlman, "Why software testing is sometimes ineffective: Two applied studies of positive test strategy." *J.Appl.Psychol.,* vol. 79, no. 1, 1994, pp. 142.

[21] H.A. Witkin, C.A. Moore, D.R. Goodenough and P.W. Cox, "Field-dependent and field-independent cognitive styles and their educational implications," *Review of Educational Research,* vol. 47, no. 1, 1977, pp. 1-64.

[22] C. Ling and G. Salvendy, "Effect of evaluators' cognitive style on heuristic evaluation: Field dependent and field independent evaluators," *International Journal of Human-Computer Studies,* vol. 67, no. 4, 2009, pp. 382-393.

[23] J. Miller and Z. Yin, "A cognitive-based mechanism for constructing software inspection teams," *Software Engineering, IEEE Transactions on,* vol. 30, no. 11, 2004, pp. 811-825.

[24] C. Chabris and D. Simons, *The invisible gorilla: And other ways our intuitions deceive us,* Broadway, 2011.

[25] A. Mack and I. Rock, *Inattentional blindness.* The MIT Press, 1998.

[26] Z.P. Fry and W. Weimer, "A human study of fault localization accuracy," *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 2010, pp. 1-10.

[27] J. Nielsen, "Finding usability problems through heuristic evaluation," *Proceedings of the SIGCHI conference on Human factors in computing systems*, 1992, pp. 373-380.

[28] J. Itkonen, M.V. Mäntylä and C. Lassenius, "The Role of the Tester's Knowledge in Exploratory Software Testing," *IEEE Transactions on Software Engineering*, Accepted Sep/2012, In press,

[29] G.J. Myers, "A controlled experiment in program testing and code walkthroughs/inspections," *Commun ACM*, vol. 21, no. 9, 1978, pp. 760-768.

[30] V.R. Basili and R.W. Selby, "Comparing the Effectiveness of Software Testing Strategies," *IEEE Trans.Software Eng.*, vol. 13, no. 12, 1987, pp. 1278-1296.

[31] J. Itkonen, M.V. Mäntylä and C. Lassenius, "Defect Detection Efficiency: Test Case Based vs. Exploratory Testing," *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on*, 2007, pp. 61-70.

[32] M. Hertzum and N.E. Jacobsen, "The evaluator effect: A chilling fact about usability evaluation methods," *Int.J.Hum.-Comput.Interact.*, vol. 15, no. 1, 2003, pp. 183-204.

[33] A. Folstad, B.C.D. Anda and D.I.K. Sjoberg, "The usability inspection performance of work-domain experts: An empirical study," *Interact Comput*, vol. 22, no. 2, 2010, pp. 75-87.

[34] A. Sears, "Heuristic walkthroughs: Finding the problems without the noise," *Int.J.Hum.-Comput.Interact.*, vol. 9, no. 3, 1997, pp. 213-234.

[35] H. Petersson, T. Thelin, P. Runeson and C. Wohlin, "Capture-recapture in software inspections after 10 years research--theory, evaluation and application," *J.Syst.Software*, vol. 72, no. 2, 2004, pp. 249-264.

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

[37] G.S. Walia, J.C. Carver and N. Nagappan, "The effect of the number of inspectors on the defect estimates produced by capture-recapture models," *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 331-340.

[38] J.C. Maldonado, J. Carver, F. Shull, S. Fabbri, E. Dória, L. Martimiano, M. Mendonça and V. Basili, "Perspective-Based Reading: A Replicated Experiment Focused on Individual Reviewer Effectiveness," *Empirical Software Engineering*, vol. 11, no. 1, 2006, pp. 119-142.

[39] C.F. Kemerer and M.C. Paulk, "The impact of design and code reviews on software quality: An empirical study based on PSP data," *IEEE Trans.Software Eng.*, vol. 35, no. 4, 2009, pp. 534-550.

[40] E. Kamsties and C.M. Lott, "An Empirical Evaluation of Three Defect-Detection Techniques," *Proceedings of the Fifth European Software Engineering Conference,* 1996, pp. 362-383.

[41] C. Andersson, T. Thelin, P. Runeson and N. Dzamashvili, "An experimental evaluation of inspection and testing for detection of design faults," *Empirical Software Engineering, 2003. ISESE 2003. Proceedings. 2003 International Symposium on,* 2003, pp. 174-184.

[42] M. Agrawal and K. Chari, "Software effort, quality, and cycle time: A study of CMM level 5 projects," *Software Engineering, IEEE Transactions on,* vol. 33, no. 3, 2007, pp. 145-156.

[43] J.D. Valett and F.E. McGarry, "A summary of software measurement experiences in the Software Engineering Laboratory," *J.Syst.Software,* vol. 9, no. 2, 2. 1989, pp. 137-148.

[44] R.D. Austin, "The effects of time pressure on quality in software development: An agency model," *Information Systems Research,* vol. 12, no. 2, 2001, pp. 195-207.

[45] R.M. Yerkes and J.D. Dodson, "The relation of strength of stimulus to rapidity of habit-formation," *Journal of Comparative Neurology and Psychology,* vol. 18, no. 5, 1908, pp. 459-482.

[46] H. Topi, J.S. Valacich and J.A. Hoffer, "The effects of task complexity and time availability limitations on human performance in database query tasks," *International Journal of Human-Computer Studies,* vol. 62, no. 3, 2005, pp. 349-379.

[47] L.S. McDaniel, "The effects of time pressure and audit program structure on audit performance," *Journal of Accounting Research,* vol. 28, no. 2, 1990, pp. 267-285.

[48] N. Bettenburg, R. Premraj, T. Zimmermann and S. Kim, "Duplicate bug reports considered harmful… really?" *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on,* 2008, pp. 337-345.

[49] W.F. Tichy, "Hints for reviewing empirical work in software engineering," *Empirical Software Engineering,* vol. 5, no. 4, 2000, pp. 309-312.

[50] S. Eldh, H. Hansson and S. Punnekkat, "Analysis of Mistakes as a Method to Improve Test Case Design," *Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on,* 2011, pp. 70-79.

[51] M. Höst, B. Regnell and C. Wohlin, "Using students as subjects—a comparative study of students and professionals in lead-time impact assessment," *Empirical Software Engineering,* vol. 5, no. 3, 2000, pp. 201-214.

[52] R. Coe, "It's the effect size, stupid: What effect size is and why it is important," *Paper presented at the Annual Conference of the British Educational Research Association*, 2002,

[53] P.D. Ellis, *The essential guide to effect sizes: Statistical power, meta-analysis, and the interpretation of research results,* Cambridge Univ Pr, 2010.

[54] K. Kelley, "The effects of nonnormal distributions on confidence intervals around the standardized mean difference: Bootstrap and parametric confidence intervals," *Educational and Psychological Measurement,* vol. 65, no. 1, 2005, pp. 51-69.

[55] J. Cohen, *Statistical power analysis for the behavioral sciences,* Lawrence Erlbaum, 1988.

[56] R. Baeza-Yates and B. Ribeiro-Neto, *Modern information retrieval,* Addison-Wesley New York, 1999.

[57] H.R. Hartson, T.S. Andre and R.C. Williges, "Criteria for evaluating usability evaluation methods," *Int.J.Hum.-Comput.Interact.,* vol. 13, no. 4, 2001, pp. 373-410.

[58] A. Dunsmore, M. Roper and M. Wood, "The development and evaluation of three diverse techniques for object-oriented code inspection," *IEEE Trans. Software Eng.,* vol. 29, no. 8, 2003, pp. 677-686.

[59] C.J.V. Rijsbergen, *Information Retrieval,* Newton, MA, USA: Butterworth-Heinemann, 1979.

[60] S. Mouchawrab, L.C. Briand, Y. Labiche and M. Di Penta, "Assessing, comparing, and combining state machine-based testing and structural testing: a series of experiments," *Software Engineering, IEEE Transactions on,* vol. 37, no. 2, 2011, pp. 161-187.

[61] P. Runeson, M. Alexandersson and O. Nyholm, "Detection of duplicate defect reports using natural language processing," *Software Engineering, 2007. ICSE 2007. 29th International Conference on,* 2007, pp. 499-510.

[62] K. Glerum, K. Kinshumann, S. Greenberg, G. Aul, V. Orgovan, G. Nichols, D. Grant, G. Loihle and G. Hunt, "Debugging in the (very) large: ten years of implementation and experience," *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles,* 2009, pp. 103-116.

[63] G. Puttick, S. Van Esch, S. Van Esch and S. Kana, *The principles and practice of auditing,* Juta & Company, 2008.

[64] R.F. Thompson, P.M. Groves, T.J. Teyler and R.A. Roemer, "A dual-process theory of habituation: Theory and behavior," *Habituation,* vol. 1, 1973, pp. 239-271.

[65] G.S. Walia, J.C. Carver and T. Philip, "Requirement error abstraction and classification: a control group replicated study," *Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on,* 2007, pp. 71-80.

[66] M.C.K. Yang and A. Chao, "Reliability-estimation and stopping-rules for software testing, based on repeated appearances of bugs," *Reliability, IEEE Transactions on,* vol. 44, no. 2, 1995, pp. 315-321.

[67] C. Wohlin, M. Höst, P. Runeson, M.C. Ohlsson, B. Regnell and A. Wesslén, *Experimentation in software engineering: an introduction,* Kluwer Academic Pub, 2000.

[68] S. Desikan and G. Ramesh, *Software testing: principles and practice,* Pearson Education India, 2006.

[69] L.C. Briand, "A critical analysis of empirical research in software testing," *Empirical Software Engineering and Measurement, 2007. ESEM 2007. First International Symposium on,* 2007, pp. 1-8.

[70] A. Bertolino, "Software testing research: Achievements, challenges, dreams," *Proceedings of the 2007 Future of Software Engineering,* 2007, pp. 85-103.

[71] S. Berner, R. Weber and R.K. Keller, "Observations and lessons learned from automated testing," *Proceedings of the 27th international conference on Software engineering,* 2005, pp. 571-579.

[72] D. Martin, J. Rooksby, M. Rouncefield and I. Sommerville, "'Good'Organisational Reasons for'Bad'Software Testing: An Ethnographic Study of Testing in a Small Software Company," *Proceedings of the 29th international conference on Software Engineering,* 2007, pp. 602-611.

[73] D.M. Rafi, K.R.K. Moses, Petersen K. and M.V. Mäntylä, "Benefits and Limitations of Automated Software Testing: Systematic Literature Review and Practitioner Survey," *7th Workshop on Automated Software Testing,* 2012, pp. 36-42.