An Empirical Assessment on Merging and Repositioning of Static Analysis Alarms

Niloofar Mansoor
University of Nebraska–Lincoln
Lincoln, Nebraska, USA
niloofar@huskers.unl.edu

Tukaram Muske
Tata Consultancy Services
Pune, India
tukarammuske@gmail.com

Alexander Serebrenik
Eindhoven University of Technology
Eindhoven, The Netherlands
a.serebrenik@tue.nl

Bonita Sharif
University of Nebraska–Lincoln
Lincoln, Nebraska, USA
bsharif@unl.edu

***Abstract*—Static analysis tools generate a large number of alarms that require manual inspection. In prior work, repositioning of alarms is proposed to (1) merge multiple similar alarms together and replace them by a fewer alarms, and (2) report alarms as close as possible to the causes for their generation. The premise is that the proposed merging and repositioning of alarms will reduce the manual inspection effort. To evaluate the premise, this paper presents an empirical study with 249 developers on the proposed merging and repositioning of static alarms. The study is conducted using static analysis alarms generated on C programs, where the alarms are representative of the merging vs. non-merging and repositioning vs. non-repositioning situations in real-life code. Developers were asked to manually inspect and determine whether assertions added corresponding to alarms in C code hold. Additionally, two spatial cognitive tests are also done to determine relationship in performance. The empirical evaluation results indicate that, in contrast to expectations, there was no evidence that merging and repositioning of alarms reduces manual inspection effort or improves the inspection accuracy (at times a negative impact was found). Results on cognitive abilities correlated with comprehension and alarm inspection accuracy.**

***Index Terms*—Static analysis, Empirical study, Manual inspection of alarms, Repositioning of alarms, C language**

I. Introduction

Static analysis tools help to automatically detect common programming errors like *division by zero* and *array index out of bounds* [1]–[6] and even certify absence of such errors in safety-critical systems [7]–[9]. However, these tools generate a large number of false positives [1], [10]–[13]. Previous studies report that there are 40 alarms for every thousand lines of code [14], and 35% to 91% of alarms are false positives [15]. Moreover, partitioning alarms into false positives and errors requires costly manual inspection [12], [13], [16]. Several studies [13], [14], [17], [18] report the large number of false positives and the cost incurred in manual inspection of alarms as the two primary reasons for the underuse of tools in practice.

Multiple techniques have been proposed to simplify manual inspection of alarms [16], [19]–[23]. We observe that, in general, even when such techniques target reduction in manual effort required to inspect the alarms, they are being evaluated based on reduction in the number of alarms, assuming that inspecting fewer alarms reduces the manual inspection effort. However, this assumption has not been validated and *a priori* is not necessarily true since inspection of different alarms might require different effort. Therefore, we consider a recently proposed technique that aims at reduction of the manual inspection effort and evaluate to what extent it improves user's performance during manual inspection of the alarms. As an example of such technique we consider *repositioning of alarms* [19], [20]. Repositioning aims at two points. First, multiple similar alarms are grouped together so that fewer alarms get inspected, and second, alarms are reported closer to the reasons for their generation so that manual inspection effort is reduced (see Section II). Hence, we investigate *to what extent repositioning of alarms improves user's performance during manual inspection of the alarms*. Since the cognitive ability [24] of developers plays a role in how they problem solve, we also seek to investigate whether there exists a relationship between cognitive tasks and tasks involving manual inspection of alarms. We perform this investigation, because, if a relationship is observed, users for manual inspection of alarms can be selected based on their performance on cognitive tasks.

To perform the two investigations discussed above, we conduct an empirical study with 249 developers using a questionnaire made available online on Qualtrics and distributed through social media and Mechanical Turk. After filtering the responses received from 1395 participants, we considered responses of 249 participants as valid and analyzed them. The empirical evaluation results indicate that, in contrast to expectations, we do not find enough evidence that the merging and repositioning of alarms reduces manual inspection effort or improves accuracy of the inspection results, and sometimes it has a negative impact. A closer look at the results suggest that the study results are inconclusive and a more detailed study needs to be performed to evaluate the premise. Furthermore, the evaluation results indicate that the participants' spatial cognition abilities (one's ability to perceive how objects relate

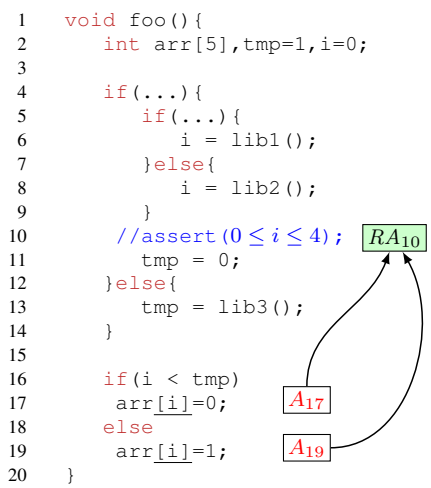


Fig. 1: A code example showcasing repositioning of alarms taken from [19].

to each other - relevant in programming) correlated with their comprehension skills and the inspection accuracy.

The paper makes the following contributions:

- An empirical evaluation conducted to validate the assumption of reduced developer effort in evaluating repositioning and merging of alarms.
- An investigation studying the relationship between cognitive tasks and manual inspection of repositioned and merged alarms.
- A complete replication package for reproducibility.

II. Background and Related Work

In this section we discuss related work in repositioning alarms. For additional details, we direct the reader to a survey of 130 studies that propose techniques for processing alarms after they are generated [24].

To overcome limitations of techniques that cluster similar alarms, repositioning of alarms has been proposed recently. Repositioning of alarms is performed with two goals in mind: (1) to reduce the number of alarms by safely merging multiple similar alarms together; and (2) to report alarms closer to the causes for their generation, so that manual inspection effort gets reduced. In the rest of the paper, we use *merging of alarms* to refer to the first repositioning goal (reducing the number of alarms by safely merging multiple similar alarms together), and *closer reporting* (or simply *repositioning*) *of alarms* to refer to the second repositioning goal (reporting alarms closer to the causes of their generation).

We illustrate the two repositioning goals by borrowing and discussing the motivating example discussed in a prior study by Muske et al. [19] shown in Figure 1. Analysis of the example code using a static analysis tool generates two alarms for *array index out of bounds* property. Such tool-generated alarms are called *original alarms*. These alarms are shown using A_{17} and A_{19}.

Repositioning these two alarms results in merging them into a single alarm and reporting the merged alarm on line 10. The alarm resulting after the repositioning is shown as RA_{10} Note that, during this repositioning, the effect of the *else* branch at line 12 is ignored, because i = 0 if the *else* branch at line 12 is taken and the alarms A_{17} and A_{19} are safe due to this value. The shown repositioning of alarms achieves both the repositioning goals: reduces the number of alarms from two to one [1], and reports alarms closer to causes of their generation. The repositioning techniques claim to reduce manual inspection effort required for A_{17} and A_{19} because,

1) the number of alarms to be inspected is reduced from two to one, and
2) during inspection of A_{17} (or A_{19}), the code that includes assignment to i on line 2 and the *else* branch on line 12 gets inspected, whereas this is not the case during inspection of RA_{10}.

III. Research Questions and Hypotheses

A. Research Questions

The goal of this research is to understand the impact of alarm repositioning on the alarms inspection performance (measured by accuracy of evaluation and speed of inspection). As explained in Section II the repositioning technique aims to achieve the effort reduction in two ways: (1) merging of alarms, and (2) reporting alarms clauser to the cause point. Therefore, we ask the following two research questions.

RQ1: Does *merging of multiple similar alarms* and representing them by fewer alarms improve alarms inspection performance?

RQ2: Does *reporting of alarms* closer to the causes of their generation improve the alarms inspection performance?

Furthermore, it has been shown in prior literature that cognitive abilities such as amount of working memory and coding skills seem to be related [25]. In other work, it has been shown that mental rotation tasks [26] are also related to problem solving [27]. In order to test if this holds for alarm inspections, we investigate whether there exists a relationship between performance on cognitive tasks and manual inspection of alarms. The findings of this investigation can help to select users for manual inspection of alarms based on their performance on cognitive tasks. The third research question is as follows.

RQ3: Is there a relationship between cognitive tasks and tasks involving manual inspection of alarms?

B. Hypotheses

Both *merging of similar alarms* (RQ1) and *closer-reporting of alarms* (RQ2) can impact performance - inspection effort (measured in time) and accuracy of the inspection results. Therefore, we answer both RQ1 and RQ2 in terms of these two

[1]When repositioning of alarms is performed, only the repositioned alarms are to be manually inspected.

parameters: reduction in manual inspection time, and gain in accuracy of inspection results. For RQ1 and RQ2, we design a separate hypothesis corresponding to each of these parameters.

As alarms inspection effort is generally measured in terms of *time taken*, we use terms *inspection effort* and *inspection time* interchangeably.

1) Alarms Merging - Inspection Time Hypothesis (MTH): The first hypothesis seeks to test if merging of similar alarms reduces alarms inspection effort. The null and alternate hypotheses follow.

MTH_0 Merging of multiple similar alarms does not cause any significant reduction in time taken to inspect the alarms.

MTH_A Merging of multiple similar alarms reduces time taken to inspect the alarms.

This hypothesis will help determine whether substituting a group of similar alarms by a fewer alarms reduces manual effort required to inspect the similar alarms. If reduction in effort is observed, similar alarms could be merged and the resulting alarms will be reported to reduce the manual inspection effort.

2) Alarms Merging - Inspection Accuracy Hypothesis (MAH): This hypothesis seeks to test if merging of similar alarms improves accuracy of manual inspection results. The null and alternate hypotheses follow.

MAH_0 Merging of multiple similar alarms does not significantly improve accuracy of manual inspection of the alarms.

MAH_A Merging of multiple similar alarms improves accuracy of manual inspection of the alarms.

This hypothesis will help determine whether substituting a group of similar alarms by a fewer alarms improves accuracy of manual inspection results. If improvement in accuracy is observed, similar alarms could be merged and reported to the user.

3) Closer reporting - Inspection Time Hypothesis (CTH): The third hypothesis seeks to test if closer-reporting of alarms reduces the code traversals performed during their inspection, thereby reducing alarms inspection effort.

CTH_0 Closer reporting of alarms does not significantly reduce time taken to inspect the alarms.

CTH_A Closer reporting of alarms reduces time taken to inspect the alarms.

This hypothesis will help determine whether closer reporting of alarms reduces manual effort required to inspect the alarms. If reduction in effort is observed, alarms can be post-processed to identify their causes and reported as close as possible to the causes.

4) Closer reporting - Inspection Accuracy Hypothesis (CAH): This hypothesis seeks to test if closer reporting of alarms improves accuracy of manual inspection results.

CAH_0 Closer reporting of alarms does not significantly improve accuracy of manual inspection of the alarms.

CAH_A Closer reporting of alarms improves accuracy of manual inspection of the alarms.

This hypothesis will help determine whether closer reporting of alarms improves accuracy of the manual inspection results. If improvement in accuracy is observed, alarms can be reported as close as possible to the causes for their generation.

5) Cognitive Tasks - Alarms Inspection Relationship Hypothesis (CogAH): This hypothesis (related to RQ3) seeks to test if a relationship between cognitive tasks and tasks involving manual inspection of alarms exists.

$CogAH_0$ There does not exist a relationship between accuracy on cognitive tasks and tasks involving manual inspection of alarms.

$CogAH_A$ There exists a relationship between accuracy on cognitive tasks and tasks involving manual inspection of alarms.

Based on this hypothesis, if the relationship between accuracy on cognitive tasks and alarms inspection tasks is observed, users performing better (more accurate) with cognitive tasks can be selected for manual inspection of alarms.

IV. Experiment Design

A. Study Overview

The goal of this controlled experiment [28], [29] is to analyze the inspection of alarms in C code for the purpose of evaluating repositioning and merging alarms from the point of view of a C developer in the context of whether or not static analysis alarms hold. Participants were asked to complete a series of comprehension and cognitive tasks. The study was approved by the university's institutional review board.

The study was hosted online on Qualtrics [30], a platform that allows building and distributing surveys. Qualtrics enables implementation of complex logic and randomization and allows customization of surveys. It also enables the survey distributor to see how much time is spent on each question in milliseconds, which we used for task duration. The survey was distributed through social media as a direct link (Link Share) and also put up as a Mechanical Turk Human Intelligence Task (HIT) [31]. On the Mechanical Turk website, a requester can post an HIT and workers can accept and complete the HIT. The requester can then check the submission and accept or reject the HIT. Accepting a HIT will automatically pay a worker for their task. Since the study was expected to take two hours, we paid the participants twice the hourly minimum wage of the state we are based in. A complete replication package[2] is available for download.

B. Tasks and Groups

We had three sets of main tasks, *cognitive*, *proficiency* and *comprehension* tasks. We also included a pre-questionnaire that asks the participants about their demographic details and a post-questionnaire asking about their their overall experience.

[2]https://osf.io/u32hj/?view_only=fe862992cef34476a2407bda6ec8b731

TABLE I: Summary of the task groups considered in the study, different treatments given, and task selection for each participant.

	Category of tasks	#Tasks	Available treatments	#Available tasks	#Tasks assigned to each participant	# of Lines in Code Per Task	Assignment Logic	Related RQs
Cognitive tasks	3D Mental Rotation Task	30	-	48	30	-	Random	RQ3
	Operation Span Task	10	-	-	10	-	Random	
Proficiency tasks	Proficiency Tasks	5	-	-	5	Between 7 - 14 lines	All, randomized	-
Comprehension tasks	Learning Task	1	No-repositioning (NR), Repositioning (R)	2 (L-NR, L-R)	1 (L-R/L-NR)	183	Random	-
	Manual Inspection of Alarms (Task Group A)	2	-	4 (A-1, A-2, A-3, A-4)	2	A-1 : 350 (1 file) A-2 : 219 (1 file) A-3 : 353 (1 file) A-4 : 166 (1 file)	Random	RQ3
	Repositioning (closer-reporting) of an Alarm (Task Group B)	2	No-repositioning (NR), Repositioning (R)	4 (B-1-NR, B-1-R B-2-NR, B-2-R)	2 (B-1-NR/B-1-R, B-2-NR/B-2-R)	B-1 : 135 (1 file) B-2 : 160 (1 file)	Random	RQ1, RQ3
	Merging of Similar Alarms (Task Group C)	2	No-merging (NM), Merging (M)	4 (C-1-NM, C-1-M C-2-NM, C-2-M)	2 (C-1-NM/C-1-M, C-2-NM/C-2-M)	C-1 : 188 (1 file) C-2-NM : 2526 (7 files) C-2-M : 1693 (3 files)	Random	RQ2, RQ3

1) Cognitive Tasks: After completing the pre-questionnaire on demographic and experience details, the participants received links and unique IDs to complete two separate cognitive tasks: 1) operation span task [32], [33] and 2) 3D mental rotation task [26]. They measure spatial recognition ability and working memory respectively. Baum et al. [25] discovered that there is an effect from working memory capacity on bug localization accuracy in code reviews. Sharafi et al. [27] also found that spatial ability and data structure manipulation are related neural tasks. We included these tasks to explore whether these correlations exist when it comes to solving our comprehension tasks. Since our study was hosted online, we used the Magpie framework [34] to build psychological experiments that run on the participants' browser. We used the open source Mental Rotation Task [35] and hosted it on a website which was linked in the survey. We also developed the operation span task using the magpie framework [36], and included the link to this task in our survey as well.

Each trial in the 3D mental rotation task shows two images of 3D objects to participants, and asks them to determine if the objects are the same or different. We gave the participants five practice trials with the answers shown to them after answering the questions, and thirty main trials to complete the task. The operation span task included three practice trials and ten main trials. In each trial of the task, the participants are asked to verify the correctness of a math equation (a distractor), and then memorize a letter that comes after the equation. After a number of equations and letters are shown, the participant will be asked to enter the letters they have memorized in a text box, in the order that they have seen them.

2) Proficiency Tasks: Subsequently, the participants were asked to answer five short C programming questions in randomized order. We included these questions to be able to exclude the participants who did not have C programming skills. Danolova et al. [37] argue that it's important to include proficiency or screening questions in surveys, specifically in surveys done using online recruitment platforms. Failure of adding proficiency questions to the experiment, will lead to inclusion of corrupt data from participants with no programming experience.

3) Comprehension Tasks: The next question presented was a learning/tutorial comprehension task. This task included an assertion inside the code for the participant to inspect and check, and we included the answer to the question. The learning task had both a repositioned and non-repositioned treatments, and one of these two treatments was randomly selected for each participant. Next, six more programming questions were presented to the participant. Each question included manually evaluating assertions placed on certain code lines. The main prompt was: *Does the assertion on the line hold?*. The participants were given the choices "Yes" and "No", and an optional text box to comment about their reasoning. Accuracy (score) of the answer is considered "correct" if the assertion evaluation is correct. Our study followed a within-subjects design [38], where all participants are exposed to all treatments albeit in different tasks to avoid learning effects. The comprehension tasks were set up and randomized in a way that each participant would see all the different treatments (repositioned/non-repositioned, merged/non-merged). Table I shows a summary of all tasks and the related research question.

For the *Manual Inspection of Alarms* task category, we designed four different tasks and two of them were randomly selected and presented to each participant. We denote these tasks as Category A tasks. For the tasks related to the *Repositioning of A Single Alarm* task category, we had two different tasks with the treatments of repositioned and non-repositioned, and each participant saw one repositioned and one non-repositioned task. We call tasks in this category category B tasks. Similarly, the C task group was for tasks related to the *Merging of Similar Alarms* category, for which we had two tasks with merged and non-merged treatment. We showed each participant one merged and one non-merged task.

For Category B and C tasks, to ensure randomization and that the participants are getting both of the treatments, we selected the first question of each category and randomized the treatment, and ensured that the participant gets the other treatment for the second task in the same category. We implemented this logic using Qualtrics' randomization and survey flow tool. An example of the randomization of comprehension tasks for a specific participant is shown in Table II. This logic ensured that each participant gets one task each for repositioned, non-repositioned, merged, and non-merged treatments. We followed the convention shown in Table II for naming the tasks with their treatments.

TABLE II: Randomized order of tasks for a sample participant

Order	Task Name
1	LearningTask-A (Non-repositioned)
2	A-4
3	A-3
4	B-1-R (Repositioned)
5	B-2-NR (Non-repositioned)
6	C-1-NM (Non-merged)
7	C-2-M (Merged)

The naming convention of the tasks is as follows. There are three different families of comprehension tasks, which are denoted by A, B and C groups. The first letter of the task name is the name of the group it belongs to. The second letter of the task name is its number within its group. The tasks with no treatment, which are non-repositioned and non-merged, are specified with NR and NM as their last characters respectively. The tasks with repositioning or merging treatments are specified with R and M as the last character. As an example, B-1-NR, is the first task of the B group without the repositioning treatment. The tasks were chosen from realistic C applications and experienced C developers in an industrial firm were consulted for feedback on the types of tasks used. Refer to our replication package for details.

C. Variables

The independent variables in this controlled experiment are repositioning and merging of alarms. Table III summarizes the experimental design. The participants each had to complete two tasks without any treatments, one task with a repositioning treatment, one task without the repositioning treatment, one task with the merging treatment, and one task without the merging treatment. The order of which the participants received a treatment or non-treatment task in each group was randomized. Thus, either the first task of the group was a treatment task, or the second one, but we ensured that all participants received one treatment task in each of the repositioning and merging groups, and one non-treatment task in each group. The dependent variables were accuracy and speed of solving the tasks. For the cognitive tasks, we measured the accuracy score of each task.

TABLE III: Experiment Overview (within-subjects design)

Goal	Study the effect of repositioning and merging alarms
Independent Variables	Repositioning Alarms Merging Alarms
Task Types (Treatments)	No treatments (manual inspection of alarms) Repositioned/Non-repositioned Merged/Non-Merged
Dependent Variables	Accuracy, Speed
Secondary Factors	Mental Rotation and Operation Span Scores

D. Participants

We had two separate but identical Qualtrics surveys set up for participants recruited through Mechanical Turk and participants recruited through Link Share. We inspected the submissions and if we observed that participants did not meet specific requirements on timing and proficiency, we excluded those participants. We set the minimum amount of time for completing all the tasks presented to a participant to 30 minutes, as our pilot study with five participants indicated 30 minutes as the minimum time to analyze and answer the questions. To confirm that we did not delete useful data by setting the cutoff time to 30 minutes, we randomly chose a subset of the entries with duration under 30 minutes. We compare the individual tasks' duration with the entries from our pilot study, and we noticed that the amount of time spent on each task is significantly lower compared to our trusted pilot participants.

Furthermore, we only kept the entries of participants who at least completed the proficiency questions and the first two comprehension tasks, as we needed the completion of at least two of the main comprehension tasks for analysis. If the participants did not go further than the learning task, their entries would be discarded, as they had not completed any of the experiment's main tasks. To further prevent including data from participants who did not make an effort on completing specific tasks, we set a cutoff duration for each individual task under analysis as well. We explain this process in Section V-A.

E. Study Instrumentation

We published the survey under the name "Assessing Static Analysis Alarms" in Mechanical Turk with the following description to give the MTurk workers an idea on the purpose of the study: "The purpose of this study is to understand how developers resolve/manually inspect static analysis alarms in the C programming language. You will be asked to complete a series of programming and cognitive tasks. It will take about two hours of your time.".

The time allotted for each worker was three hours to prevent the participants from running out of time, and to account for unexpected events such as internet outages. The study was published in batches and for 20 workers at a time, to easily

manage accepting and rejecting submissions. After 20 workers would complete the study, we examined their submissions and decided on whether to accept or reject them. The survey was published to workers with the criteria of being employed in Software and IT services industry. After accepting the HIT, the workers were shown the Qualtrics survey link and they were asked to enter the ID that was generated for them upon completing the Qualtrics survey into a text box. They were also asked to enter their MTurk worker ID on the first page of the Qualtrics survey, so that we could ensure the validity of the submissions on the survey. We also recruited participants through sharing the survey link on social media and with computer scientists and developers we knew as friends or coworkers. The participants who partook in the study through following the link had to email the survey administrator with their Qualtrics ID to get their incentive.

We tailored the survey in a way that the code would look similar to what the participants see in their IDEs while working on software development tasks. We used PrismJS [39] to highlight the syntax of the code shown to participants. Since some of the tasks required traversing multiple C files, we created a tabbed view that included each file on each tab inside the Qualtrics questions (see replication package for the figure of the view), but unfortunately the Qualtrics customization did not allow us to insert such JavaScript code into the tasks. We worked around this limitation by hosting the tabbed view including all the necessary files for each question on another website, and embedding the page into the corresponding Qualtrics questions.

V. Experimental Results

A. Data Cleaning and Pre-processing

Initially we had 1254 entries into our Link Share Qualtrics survey, and 141 entries into our Mechanical Turk Qualtrics survey. Inspection of the data indicated that a number of participants did not substantially complete the survey. At first, we deleted the entries from participants who did not progress beyond the learning task, spent less than 30 minutes on the entire survey (as explained in section IV-D) and they did not complete any of the main tasks. Subsequently, we checked the data for obvious spam data (e.g. copy/paste comments and keysmashing), and deleted records from participants who spent less than 20 seconds on each question.

After performing the explained steps, we were left with 151 entries from our Link Share Qualtrics survey, and 98 entries from our Mechanical Turk Qualtrics survey. In our pre-questionnaire, we asked the participants demographic and experience details. Table IV summarizes these demographic and experience details of the participants from the two surveys.

Among the remaining submissions, we analyzed the tasks separately, since there might have been participants who completed some of the tasks and stopped working on the survey. To get a better idea on how much time should be spent on each task, and to define a cutoff to delete the entries in which the participants were too fast or too slow to solve the task, we put together a list of fifteen participants whom we trusted spent significant time on the study. These were students or programmers that the authors personally reached out to and asked them to complete the study. We separated the six tasks presented to the participants (j = number of tasks, $1 \leq j \leq 6$) and we found the minimum time spent on each task among the trusted participants ($Trusted_j$). We then excluded the entries where task j took less time than $Trusted_j$. Subsequently, we used the interquartile range method to detect the outliers who spent an unusually long time working on each task.

In the last step, we also considered the proficiency in the C programming language by setting a criteria on the proficiency questions. From the remaining entries, we only kept the tasks from the participants who received a score greater and equal to 2 out of 5 from the proficiency questions. Figure 2 shows the differences between the level of experience of the participants of Link Share vs Mechanical Turk. Most of the participants from Link Share and Mechanical Turk think that they are

TABLE IV: Summary of demographic and experience details from Link Share and Mechanical Turk participants

	Choices	Link Share	Mechanical Turk
Age	19-23	33	3
	24-28	55	49
	29-33	37	29
	34-38	19	12
	39-43	2	1
	44-48	3	2
	49+	2	2
Gender	Woman	40	45
	Man	107	52
	Non-binary	2	1
	Undisclosed	2	-
Programming Experience Compared to Peers	Less experienced	49	17
	Equally experienced	80	63
	More experienced	22	18
Years of Programming Experience	Less than 1 year	34	9
	Between 1 and 3 years	50	41
	Between 3 and 5 years	37	34
	More than 5 years	30	14
Usage of Static Analysis Tools	Yes	113	52
	No	38	46
Frequency of Fixing Bugs	A couple times a year	33	17
	Every month	54	43
	Every week	44	19
	Everyday	20	19
Most Familiar IDE	Visual Studio	70	60
	Eclipse	25	26
	NetBeans	18	12
	Intellij	27	-
	Other	11	-
Years of Experience in Industry	0-1	59	7
	2-4	69	73
	5+	23	18

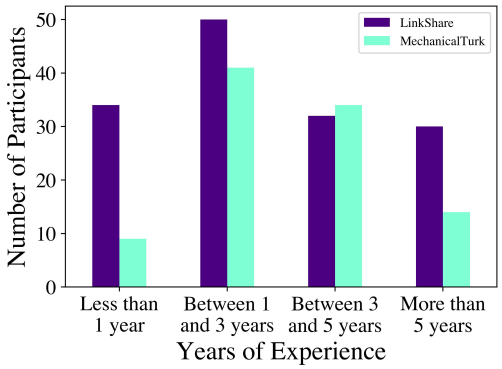


Fig. 2: Years of experience in programming for Link Share and Mechanical Turk survey participants

TABLE V: Number of entries for each task

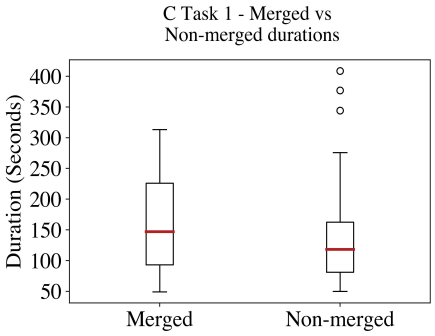
	Link Share	Mechanical Turk	Overall
A Tasks	34	19	53
B-1	31	20	51
B-2	30	12	42
C-1	27	14	41
C-2	23	21	44

equally experienced to their peers. The Mechanical Turk and Link Share participants had different levels of experience in programming and also years of industry experience. Due to these differences, we decided to analyze the differences between the two groups instead of combining the entries and analyzing the combined data in search of the answers for our research questions. Table V shows how many entries were left after cleanup of the data for each task, in the LinkShare and Mechanical Turk surveys.

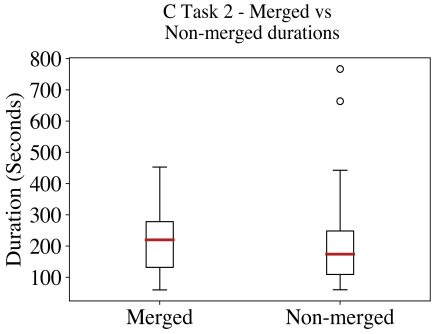
B. RQ1 Results : Merging and Performance

We first examined the differences between the speed and score obtained by participants from Mechanical Turk and LinkShare on C-1 and C-2. We used ANOVA on the duration and score of participants for each of the tasks, and calculated the effect size using Cohen's *d*. We did not find differences between the groups, so we do not distinguish between LinkShare and Mechanical Turk participants' submissions for these tasks.

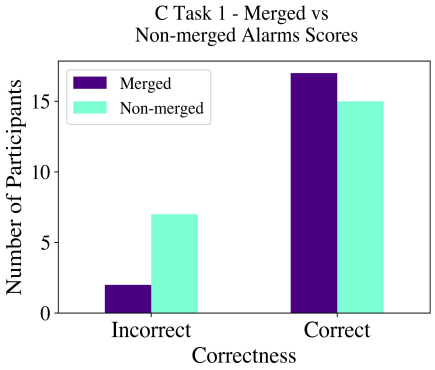
Next, we compared the duration and scores of participants who worked on the merged and non-merged treatments for each task. Figure 3 shows the box plot of the time participants worked on each treatment, and the scores of the participants who worked on the different treatments. For Task C-1, we first ran the Shapiro-Wilk test on the two merged and non-merged groups' speed to determine the normality of the data distribution. We found that data was not normally distributed. We then ran the Mann-Whitney test to see if the differences between the two merged and non-merged treatments are meaningful. The test showed that there are no significant differences between the speed of the participants working on merged and non-



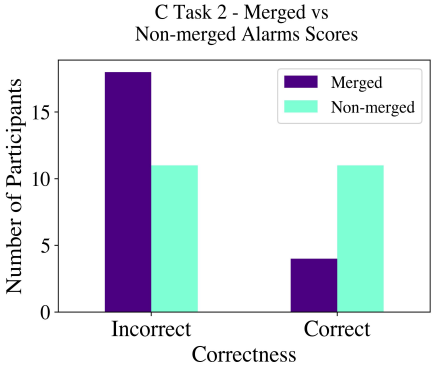
(a) Durations of treatments of Task C-1



(b) Durations of treatments of Task C-2



(c) Accuracy of answers for treatments of Task C-1



(d) Accuracy of answers for treatments of Task C-2

Fig. 3: Comparison of different treatments in the C task group

merged treatments (Merged treatment: N = 19, Non-Merged treatment: N = 22, p = 0.504, d = 0.1035).

To examine the effect of the treatment on the accuracy of answers, we gave the score of 0 or 1 based on the accuracy of the answers for the task, and treated them as categories of "correct" and "incorrect". We ran the Fisher's Exact test to determine if there are non-random associations between two categorical variables. The test showed that there is not enough evidence to claim there's any associate between accuracy of answers and merging of alarms (p = 0.14). We repeated the same steps for the speed of the participants receiving different treatments on Task C-2 (Merged treatment: N = 22, Non-Merged treatment: N = 22). We found no significant differences between the speed (Mann-Whitney test p-value = 0.391, d = 0.004) and no association between accuracy of answers and merging of alarms (p = 0.0546).

We could not reject the null hypotheses MTH_0 and MAH_0, indicating lack of evidence that merging multiple similar alarms improves the accuracy and inspection effort.

C. RQ2 Results: Repositioning and Performance

To investigate performance on tasks of group B, we looked at the accuracy and speed of completing the tasks. Once again, we examined the differences between the speed and accuracy score from Mechanical Turk and LinkShare participants who completed the tasks, to see if the groups have any effect on the speed and accuracy of tasks. Using ANOVA on the Tasks B-1 and B-2 durations and scores shows there are differences between the Mechanical Turk and LinkShare durations for task B-1 (ANOVA p = 0.036, d = 0.61), but not on task B-2 (ANOVA p = 0.055, d = 0.674). Based on these results, we decided to analyze the effect of repositioning on the Mechanical Turk and LinkShare task B-1 entries separately, but for task B-2 we merged the entries from the two sources.

For tasks B-1 LinkShare, B-1 Mechanical Turk, and B-2, we looked at the effect of repositioning of alarms on duration and score. See Figure 4. For all groups of tasks, we first ran the Shapiro-Wilk test to test whether the distribution of data was normal. We found that the repositioned and non-repositioned tasks' durations were normally distributed on task B-1 LinkShare, but not on the other groups. To investigate the effect of repositioning on the speed of problem solving, we used the T-Test or Mann-Whitney test based on the normality of data. Table VI shows the results of the statistical tests, and that none of the tests showed any significant effect on the speed from repositioning. We used the Fisher's Exact test to explore whether the repositioning treatment has an effect on the accuracy scores. We could not find any evidence that this effect existed based on the Fisher's Exact test (B-1 LinkShare: p = 0.293, B-1 MTurk: p = 0.255, B-2: p = 1)

We cannot reject the null hypotheses CTH_0 and CAH_0, indicating lack of evidence that repositioning (closer-reporting) of alarms increases accuracy and speed of manual inspection.

D. RQ3 Results: Cognitive Tasks and Performance

For RQ3, we looked at the relationship between the comprehension tasks accuracy score (C_{sco}) of the participants on the comprehension tasks to the scores they obtained on the two cognitive tasks. To find out which cognitive task entries were useful, we found the participants with proficiency score $\geq$ 2, who completed at least one comprehension task based on our cutoff criteria. The results were the entries of 122 participants. Since the participants were asked to complete six comprehension tasks, we calculated their score on the survey out of six. We gave a score of zero for each task that the participants did not attempt to complete. (N = 122, $Mean_{C_{sco}} \pm SD$ = 1.93±1.22)

TABLE VI: Results of statistical tests for Task group B

	Test	p	Effect Size	Repositioned Group N	Non-repositioned Group N
B-1 LinkShare	T-Test	0.889	-0.050	13	18
B-1 MTurk	Mann-Whitney	0.623	-0.104	8	12
B-2	Mann-Whitney	0.989	-0.168	26	16

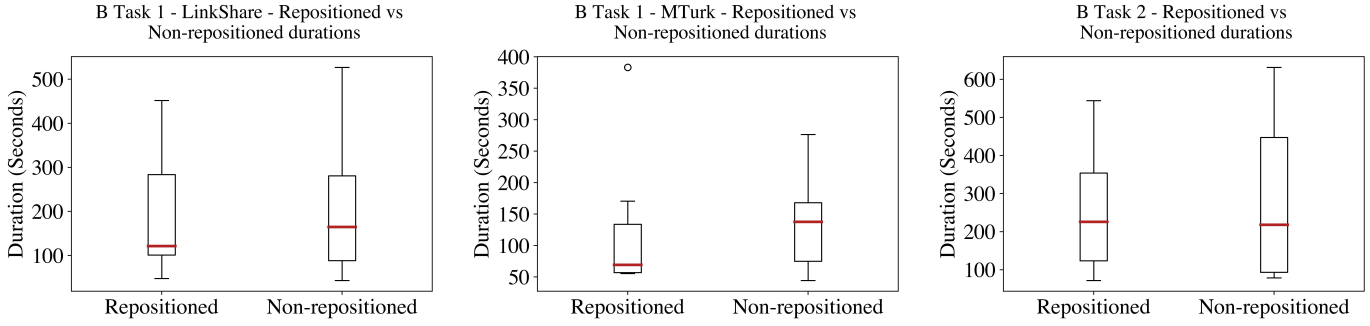
We then calculated the scores on the cognitive tasks. In the 3D Mental Rotation task, the participants saw five practice trials and thirty main trials. In each trial, they saw two images of two 3 objects side by side, and they decided if the objects were the same or different. The participants would get one point for each correct answers on the main tasks. Thus, the scores of the participants (denoted by MRT$_{sco}$) are calculated out of 30 (N = 111, $Mean_{MRT_{sco}} \pm SD$ = 22.13±6.17). In the Operation Span (OSPAN) Task, we calculated the score based on the operation span task MIS scoring method presented by Lammert et al. [40]. The MIS method takes into account the performance on both the letter recall and the correctness of math equation check distractor tasks. For the operation span task, we had ten main tasks for the participants, and the MIS score (denoted by OSPAN$_{sco}$) for each task is out of 1, leading to the overall highest possible score being 10 ($N = 94$, $Mean_{OSPAN_{sco}} \pm SD$ = 6.077±1.66).

To find the relationship between the cognitive scores and C_{sco}, we used Pearson's correlation. We found that there was a moderate positive correlation between the MRT_{sco} and C_{sco}, $r = 0.431$, $p < 0.005$, with mental rotation task score explaining 18% of the variation in comprehension tasks scores. However, we did not find a correlation between the $OSPAN_{sco}$ and C_{sco}. ($r = 0.189$, $p = 0.069$)

Our findings for RQ3 show evidence that can reject the null hypothesis $CogAH_0$, indicating there is a relationship between a programmers' spatial abilities and their problem solving skills.

E. Post Questionnaire Results

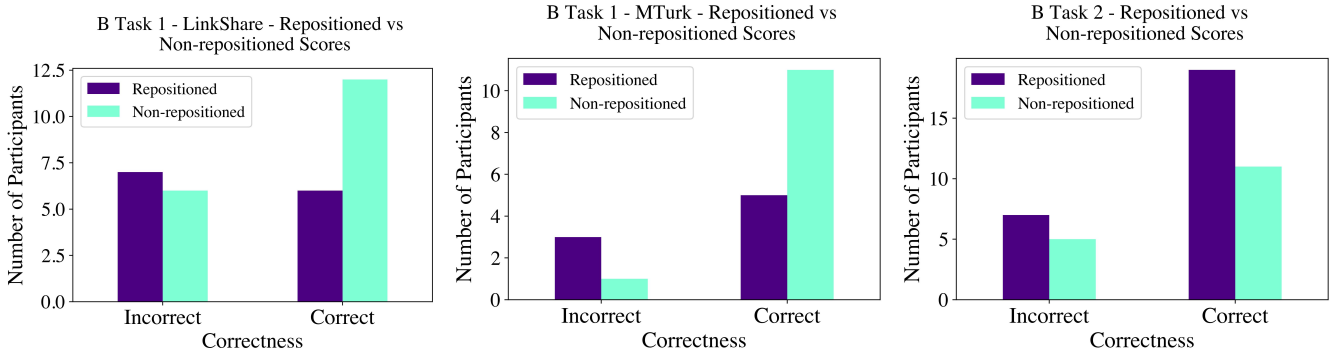
Overall, most participants found the study somewhat difficult. Breaking this down further, out of the 122 participants with proficiency test scores $\geq$ 2 who had at least completed one task after the learning task, 21 participants believed that the survey was "Extremely difficult", 48 participants thought it was "Somewhat difficult", 24 said it was "Neither easy nor



(a) Durations of treatments of Task B-1 - LinkShare

(b) Durations of treatments of Task B-1 - Mechanical Turk

(c) Durations of treatments of Task B-2 (LinkShare and MTurk combined)



(d) Accuracy of answers for treatments of Task B-1 - LinkShare

(e) Accuracy of answers for treatments of Task B-1 - Mechanical Turk

(f) Accuracy of answers for treatments of Task B-2

Fig. 4: Comparison of different treatments in the B task group

difficult", 10 people believed it was "Somewhat easy", and 4 believed that it was "Extremely easy".

F. Threats to Validity

Internal Validity: Confounding factors such as proficiency in the C programming language and the online nature of the study are threats to the internal validity of the study. We tried to mitigate the lack of proficiency in C by asking the participants to complete a proficiency test and excluding the participants who could not achieve a proficiency score ≥ 2. Since we could not monitor the participants due to the online nature of the study, we tried to use duration as an indicator of attention while working on the tasks. However, the cutoff time might have affected the number of entries we had for each task. Additionally, when we split participants into blocks of < 5 years and ≥ 5 years of programming experience, we did not see any significant differences in the results.

External Validity: The comprehension tasks chosen for this study were taken from open source software and the realistic nature of the tasks makes them a good representative of what software developers work on. In terms of population validity, the participants had varying levels of expertise and we had more participants with less than five years of experience compared to participants with more than five years of experience (Figure 2). This can possibly be a threat to the generalizability of the results, especially when trying to understand whether applying these methods for experienced programmers can make a difference in their work.

Construct Validity: Our dependent variables were chosen carefully to make sure they represent what we seek to measure. Due to the online nature of the study, we measured the accuracy and the speed of participants while working on comprehension tasks. These measures can give us some ideas about whether the repositioning or merging of the alarms had an effect on inspection time and difficulty of problem solving.

Conclusion Validity: With respect to conclusion validity, we ensured to test all assumptions for the statistical tests used. We used the unpaired Mann-Whitney test to compare the averages of the two independent groups in the study, which is suitable for small samples that are not normally distributed. For the normally distributed samples, we used the T-Test.

VI. Discussion and Implications

This section discusses possible reasons for no reduction in manual inspection effort.

A. Impact on Manual Effort Reduction

When it comes to manual effort reduction several possible explanations can be provided for the inconclusive results.

1) Indeed, there is no reduction in manual inspection effort due to the alarms merging and repositioning.
2) There is reduction, but it is negligible when compared with the effort required to inspect the repositioned alarms.

3) There is significant reduction, however the study setup and design is not able to capture it due to sample size.

Following we discuss these possibilities in detail.

1) Possibility 1 - No reduction: Based on our experience with working on manual inspection of alarms, and also as argued in the original paper introducing repositioning of alarms [19], we expected reduction in inspection effort, as the merging of alarms reduces the numbers to be inspected, and reduces the code traversals to be performed. In our own manual analysis of those tasks, we found that there is some code that gets skipped due to the alarms merging and repositioning. Therefore, based on the theoretical grounds we consider possibility 1 to be unlikely.

2) Possibility 2 - Negligible reduction: We analyze our study setup and manual inspection process in practice to identify differences in the process as those could also be the reasons for not observing significant improvement in the inspection effort. We observe the following:

(1) *Number of alarms reported matter:* In our tasks, only the alarms that are merged are considered. However, in practice, there are thousands of alarms [14]. When these alarms are reported to the user, it is not guaranteed that similar alarms will get reported together. Thus, their manual inspection can be performed at different times, and the context switch happens when user switches from one alarm to another alarm. In our tasks, there was no context switches impact as those alarms were from the same code locations (functions).

(2) *Reduced effort is negligible as compared to the remaining effort:* In general, inspecting alarms is a time consuming process as it requires to traverse the code to figure out values for variables involved. This is true for original alarms as well the alarms resulting after merging and repositioning. In our manual inspection of those merged and repositioned alarms, we found that the time required to inspect is considerable, and this effort dominates the effort that gets reduced.

3) Possibility 3 - Significant reduction: This possibility relates to the reduction being present but our study not being able to detect it due to the study setup. It could be due to the following reasons:

(1) *Small number of participants:* We had a large number of participants taking the study. However, the number of submissions left after preprocessing and cleaning the results, for each of the Link Share and Mechanical Turk categories is small compared to the developer population.

(2) *Within-subjects study:* Since a participant cannot complete a same task with and without a treatment due to learning effects, we designed the study as within-subjects. As the experience and proficiency with programming languages affect the inspection time, it is possible that the reduced time is nullified.

B. Impact on Inspection Accuracy Improvement

As discussed earlier (Section V), the merging and repositioning of alarms does not improve the inspection accuracy. Our analysis to understand the reasons for this resulted in the following possibilities:

(1) *Assumptions about the code:* Every user has different assumptions about APIs and the code encountered during the inspection process, e.g., whether the arguments passed to *main* function can take null value. Therefore, it is possible that participants made different assumptions for the same code during the inspection process and ultimately reached different conclusions.

(2) *Code context is different for repositioned alarms:* The feedback given by some of the participants during the pilot stage of the study included that the variables present in the assertions were not found in the code surrounding to it, and it confused some of the participants. These cases arise as repositioning reports alarms at different locations than the locations where the error can occur (the involved variables are actually used). Considering this could be a reason, a more detailed investigation is required.

Confirming the premise of impact of merging and repositioning of alarms offers a variety of new insights and opportunities to design new techniques to reduce code traversals performed during the alarms inspection, and thus simplify the inspection process.

C. Cognitive Skills Effect on Accuracy

The results for RQ3 show that comprehension tasks' scores are more affected by the participants' spatial cognition skills (as measured by the 3D Mental Rotation task) compared to working memory (as measured by Operation Span Task). This relationship can be explained by the nature of the comprehension tasks, which mostly asked the participants to explore the state and value of a variable throughout the task, and might have led them to create a map of the variables in their mind to follow to answer the questions. The tasks were not memory intensive tasks, and they did not require the participants to memorize the values of different variables or states at a time. This could explain why we could not see a relationship between working memory and accuracy of the comprehension tasks.

VII. Conclusions and Future Work

The paper investigates the effect of merging and repositioning of alarms on their inspection time and accuracy. We designed and conducted a Qualtrics within-subjects study for this purpose. The findings from our sample, do not provide enough evidence indicating that repositioning and merging of alarms have an effect on the alarms' inspection time and accuracy. However, our analysis showed that the participants' spatial cognitive abilities correlated with their comprehension skills and accuracy. For future work, we plan to conduct the same study using an eye tracker to get more detailed and fine-grained data about the effects of alarms merging and repositioning, and understand the inspection process to identify other factors that can help reduce the inspection effort.

Acknowledgment

This work has been partly funded by the National Science Foundation under grant number CCF 18-55756.

REFERENCES

[1] C. Sadowski, J. Van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *International Conference on Software Engineering*. IEEE, 2015, pp. 598–608.

[2] L. N. Q. Do, J. Wright, and K. Ali, "Why do software developers use static analysis tools? a user-centered study of developer needs and motivations," *IEEE Transactions on Software Engineering*, 2020.

[3] N. Ayewah, W. Pugh, J. D. Morgenthaler, J. Penix, and Y. Zhou, "Evaluating static analysis defect warnings on production software," in *Workshop on Program Analysis for Software Tools and Engineering*. ACM, 2007, pp. 1–8.

[4] A. Bessey, K. Block, B. Chelf, A. Chou, B. Fulton, S. Hallem, C. Henri-Gros, A. Kamsky, S. McPeak, and D. Engler, "A few billion lines of code later: using static analysis to find bugs in the real world," *Communications of the ACM*, vol. 53, no. 2, pp. 66–75, 2010.

[5] A. Venet, "A practical approach to formal software verification by static analysis," *ACM SIGAda Ada Letters*, vol. 28, no. 1, pp. 92–95, 2008.

[6] N. Ayewah and W. Pugh, "The Google FindBugs fixit," in *International Symposium on Software Testing and Analysis*. ACM, 2010, pp. 241–252.

[7] A. Kornecki and J. Zalewski, "Certification of software for real-time safety-critical systems: state of the art," *Innovations in Systems and Software Engineering*, vol. 5, no. 2, pp. 149–161, 2009.

[8] G. Brat and A. Venet, "Precise and scalable static program analysis of NASA flight software," in *Aerospace Conference*. IEEE, 2005, pp. 1–10.

[9] E. Denney and S. Trac, "A software safety certification tool for automatically generated guidance, navigation and control code," in *Aerospace Conference*. IEEE, 2008, pp. 1–11.

[10] X. Rival, "Abstract dependences for alarm diagnosis," in *Asian Symposium on Programming Languages and Systems*. Springer, 2005, pp. 347–363.

[11] R. Mangal, X. Zhang, A. V. Nori, and M. Naik, "A user-guided approach to program analysis," in *Joint Meeting on Foundations of Software Engineering*. ACM, 2015, pp. 462–473.

[12] I. Dillig, T. Dillig, and A. Aiken, "Automated error diagnosis using abductive inference," in *Conference on Programming Language Design and Implementation*. ACM, 2012, pp. 181–192.

[13] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *International Conference on Software Engineering*. IEEE, 2013, pp. 672–681.

[14] M. Beller, R. Bholanath, S. McIntosh, and A. Zaidman, "Analyzing the state of static analysis: A large-scale evaluation in open source software," in *International Conference on Software Analysis, Evolution, and Reengineering*. IEEE, 2016, pp. 470–481.

[15] S. Heckman and L. Williams, "A systematic literature review of actionable alert identification techniques for automated static code analysis," *Information and Software Technology*, vol. 53, no. 4, pp. 363–387, 2011.

[16] T. Muske, A. Baid, and T. Sanas, "Review efforts reduction by partitioning of static analysis warnings," in *Source Code Analysis and Manipulation (SCAM), 2013 IEEE 13th International Working Conference on*, Sept 2013, pp. 106–115.

[17] L. Layman, L. Williams, and R. S. Amant, "Toward reducing fault fix time: Understanding developer behavior for the design of automated fault detection tools," in *International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2007, pp. 176–185.

[18] M. Christakis and C. Bird, "What developers want and need from program analysis: An empirical study," in *International Conference on Automated Software Engineering*. ACM, 2016, pp. 332–343.

[19] T. Muske, R. Talluri, and A. Serebrenik, "Repositioning of static analysis alarms," in *International Symposium on Software Testing and Analysis*. ACM, 2018, pp. 187–197.

[20] ——, "Reducing static analysis alarms based on non-impacting control dependencies," in *Asian Symposium on Programming Languages and Systems*. Springer, 2019, pp. 115–135.

[21] W. Lee, W. Lee, and K. Yi, "Sound non-statistical clustering of static analysis alarms," in *International Conference on Verification, Model Checking, and Abstract Interpretation*. Springer, 2012, pp. 299–314.

[22] D. Zhang, D. Jin, Y. Gong, and H. Zhang, "Diagnosis-oriented alarm correlations," in *Asia-Pacific Software Engineering Conference*. IEEE, 2013, pp. 172–179.

[23] W. Lee, W. Lee, D. Kang, K. Heo, H. Oh, and K. Yi, "Sound non-statistical clustering of static analysis alarms," *ACM Transactions on Programming Languages and Systems*, vol. 39, no. 4, pp. 1–35, 2017.

[24] T. Muske and A. Serebrenik, "Survey of approaches for postprocessing of static analysis alarms," *ACM Computing Survey*, vol. 55, no. 3, Feb 2022.

[25] T. Baum, K. Schneider, and A. Bacchelli, "Associating working memory capacity and code change ordering with code review performance," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1762–1798, 2019.

[26] S. G. Vandenberg and A. R. Kuse, "Mental rotations, a group test of three-dimensional spatial visualization," *Perceptual and Motor Skills*, vol. 47, no. 2, pp. 599–604, 1978, pMID: 724398. [Online]. Available: https://doi.org/10.2466/pms.1978.47.2.599

[27] Z. Sharafi, Y. Huang, K. Leach, and W. Weimer, "Toward an objective measure of developers' cognitive activities," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 30, no. 3, pp. 1–40, 2021.

[28] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[29] K. Stol and B. Fitzgerald, "The ABC of software engineering research," *ACM Trans. Softw. Eng. Methodol.*, vol. 27, no. 3, pp. 11:1–11:51, 2018. [Online]. Available: https://doi.org/10.1145/3241743

[30] "Qualtrics xm - experience management software," May 2022. [Online]. Available: https://www.qualtrics.com/

[31] "Amazon mechanical turk," May 2022. [Online]. Available: https://www.mturk.com/

[32] N. Unsworth, R. P. Heitz, J. C. Schrock, and R. W. Engle, "An automated version of the operation span task," *Behavior research methods*, vol. 37, no. 3, pp. 498–505, 2005.

[33] A. R. Conway, M. J. Kane, M. F. Bunting, D. Z. Hambrick, O. Wilhelm, and R. W. Engle, "Working memory span tasks: A methodological review and user's guide," *Psychonomic bulletin & review*, vol. 12, no. 5, pp. 769–786, 2005.

[34] "Magpie experiments framework," June 2022. [Online]. Available: https://magpie-experiments.org/

[35] "Mental rotation task," June 2022. [Online]. Available: https://github.com/magpie-ea/magpie-mental-rotation

[36] "Operation span task," June 2022. [Online]. Available: https://github.com/niloofarmansoor/magpie-ospan

[37] A. Danilova, A. Naiakshina, S. Horstmann, and M. Smith, "Do you really code? designing and evaluating screening questions for online surveys with programmers," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 537–548. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00057

[38] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012.

[39] "Prismjs," May 2022. [Online]. Available: https://prismjs.com/

[40] M. Lammert, F. Morys, H. Hartmann, L. Janssen, and A. Horstmann, "MIS–a new scoring method for the operation span task that accounts for math, remembered items and sequence," 2019, psyArXiv 10.31234/osf.io/ue3j8.