AN EMPIRICAL ASSESSMENT OF FORMAL MODELS AND STATIC ANALYSIS ALARMS IN THE CONTEXT OF DEFECT DETECTION

by

Niloofar Mansoor

A DISSERTATION

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfillment of Requirements

For the Degree of Doctor of Philosophy

Major: Computer Science

Under the Supervision of Professor Bonita Sharif

Lincoln, Nebraska

December, 2022

AN EMPIRICAL ASSESSMENT OF FORMAL MODELS AND STATIC ANALYSIS ALARMS IN THE CONTEXT OF DEFECT DETECTION

Niloofar Mansoor, Ph.D.

University of Nebraska, 2022

Adviser: Professor Bonita Sharif

Software developers work with different programming languages and tools in their careers. Software maintenance is one of the essential parts of a developer's job. It includes locating and correcting defects in software systems. Understanding how developers comprehend programs and their process of finding and fixing bugs is essential for creating better tools to improve their productivity and workflow. It also gives us insight into their cognitive processes while working on these tasks, which can help build theories for teaching and learning programming. This dissertation presents three studies on assessing formal models and static analysis alarms in the context of defect detection. The first is an online empirical study on the effects of experience on bug fixing tasks in the Alloy specification language, a lightweight formal language with accompanying software that performs bounded model checking to facilitate verifying properties of software models. The study found that non-novices performed 54% better on the Alloy tasks compared to novices, and participants performed better on syntactic tasks compared to semantic tasks. Insights from the participants' problem solving patterns are presented. The second and third studies investigate the effects of repositioning and merging static analysis alarms used to detect software defects on task accuracy and developers' visual effort. The data for the first study was collected through an online survey, and in the second study, eye tracking was used to collect the participants' gazes while working on the tasks. Each study was done

independently, with no overlap between participants. Both studies' results indicate that the repositioning and merging treatment do not affect accuracy. However, the eye tracking study results indicate that there is a significant difference in visual effort with the repositioning and merging of alarms as it draws the focus and attention of the developers to the most important parts of the program to solve tasks. The impact of these results on languages and tools to assist developers in defect detection is discussed.

DEDICATION

To my parents, Fereshteh Naeimi and Abdolhossein Mansour,

for their steadfast support, encouragement, and endless love.

ACKNOWLEDGMENTS

I would like to sincerely thank my advisor, Dr. Bonita Sharif, for her constant support throughout my journey as a graduate student. I appreciate all the guidance and encouragement she has given me, and I thank her for believing in me. Her conviction, determination, and her ability to overcome tough circumstances are inspiring to me.

I'd like to express my gratitude to my committee members, Dr. Witawas Srisa-an, Dr. Brittany Duncan, and Dr. Michael Dodd. Their valuable feedback on my research helped me immensely in improving my dissertation.

I would like to extend my thanks to everyone who spent time participating in my studies. I would not have been able to conduct this research without their effort.

I thank all my friends and family members for their support and kindness. I especially thank my friend Masoomeh Hajizadeh Oghaz for being with me every step of my graduate school journey and being understanding and helpful throughout everything. I am grateful for my cat, Tommy, for providing laughter, entertainment, and distraction in tough times.

And finally, I express my deepest gratitude to my parents, without whom I would have never been able to complete my PhD. Thank you for always believing in me, for listening to all my graduate school stories, remembering all the details, and for your unwavering support and love. My parents have taught me that no matter how difficult things get, I can persevere, get through, and succeed. That is the most valuable lesson of all.

PREFACE

The results presented in Chapter 3 are published in 2022 International Working Conference on Source Code Analysis and Manipulation (SCAM) [1].

Some of the content discussed in the dissertation has been published in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) [2] and in the Workshop on Advances in Human-Centric Experiments in Software Engineering (HUMAN 2022) [3].

The results presented in chapters 2 and 4 are expected to be included in two separate manuscripts that are currently in preparation for publication under the lead of Niloofar Mansoor.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the growing number of programming languages and technologies, developers work with different programming languages and tools throughout their careers. They learn new languages and use new tools built specifically for some of their needs (such as domain-specific languages or tools) or languages with new and easier to use features that help facilitate everyday tasks. It is a critical skill to be able to build on existing expertise and knowledge and learn new programming languages and work with new tools as needed. One of the software engineering tasks on which developers spend a lot of time is bug fixing. Bug fixing deals with detecting faults in a software system and finding a solution to correct that fault. It is difficult and costly to find and fix bugs in software. Thus, developers have created many methods and tools to facilitate this process [4, 5]. Software testing [6, 7], static code analysis [8–12], and formal verification [13] are some of the different methods with different theoretical bases and varying complexity levels that come with various tools to help developers detect and patch bugs. These methods are complementary to one other: program testing is a method that is used to show the presence of bugs, but it cannot prove their absence [14] whereas formal verification can prove the correctness of programs in regards to a certain specification. While a large number of these tools are available, it is also

important to empirically study the usage of these tools and program comprehension of developers while utilizing these tools [15, 16].

Studying how developers work on different types of tasks can provide interesting insights into their skills and their thought process, which can help provide better methods for teaching and learning new languages and introducing new tools. Methods such as interviews [17, 18], observation of software repository data [19–21] and collection of biometric data such as eye tracking data to [22–28] have been used to study the processes behind finding and patching defects. Most of the empirical studies are performed on tasks in popular languages such as Java, Python, and C++ and more well-known static analysis tools such as Lint, IntelliJ, and PMD. The more tools, methods, and languages that are empirically studied add to a large body evidence that informs the developers about how to improve the tools and languages.

In this dissertation, I leverage methods from empirical software engineering and cognitive sciences to understand developers' program comprehension and their cognitive processes while they are working on bug detection and fixing tasks. Empirical studies on comprehension patterns of developers while working on a varied set of software tasks such as bug fixes and verification of static analysis alarms are presented, with the common goal of understanding how the developers approach finding defects in software specifications or code. These empirical studies are designed using a combination of online questionnaires and biometric equipment (eye tracking equipment). The eye tracking biometric measures provide fine-grained details on what areas in code developers look at as they work. The data acquired from these methods can help add insight into the thought process behind solving the comprehension tasks [29, 30].

In the studies presented here, developers work on program comprehension tasks in the context of defect detection and correction and a set of established cognitive tasks with the aim of determining whether there is indeed a relationship between the

abilities these tasks measure and domains on performance. Mental Rotation [31] and Operation Span [32] tasks are designed to measure spatial ability and working memory. Spatial ability is the ability to recognize the relationships between objects in space, and mentally manipulating those objects. This ability, often measured by mental rotation tasks, is well-researched in psychology literature [33–35], and spatial abilities are one of the factors positively affecting mathematical abilities [36–38]. Working memory is a part of human memory responsible for storing short-term information needed for ongoing cognitive processes. Working memory can be measured by complex span tasks [39]. We used the operation span task to measure working memory in developers. Prior research has shown the relationship of working memory with all types of mathematics skills, such as whole-number calculations, word problem-solving, and algebra [40, 41]. Spatial ability and working memory are studied in relation to programming abilities as well. Krueger et al. [42] found that the task of code writing activates the regions of the brain that are associated with working memory and spatial cognition. Bergersen et al. [43] found that working memory, mediated through programming knowledge, influenced programming performance, and Baum et al. [44] found a moderate correlation between working memory capacity and ability to find defects in software. Sharafi et al. [45] found that spatial ability and data structure manipulation are correlated. We chose to test the working memory and spatial cognition abilities of our participants to see if we can see find a relationship between these studied cognitive tasks and the specific defect detection tasks in our studies.

Figure 1.1 presents an overview of the studies. These three studies investigate one overall theme, which is the empirical study of developer comprehension in defect detection tasks. Two different mediums (data collection methods) were chosen for containing the defect detection tasks, and two sets of separate tasks were designed for

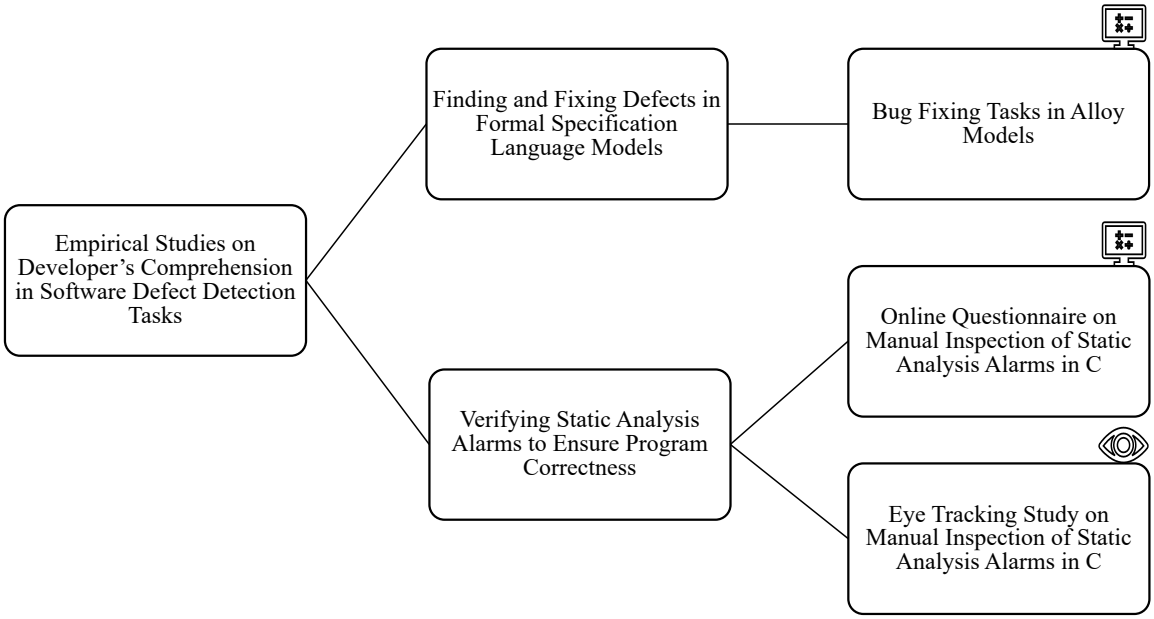


Figure 1.1: Overview of studies

each study. The goal was to explore finding and fixing bugs in a formal specification language model and using the corresponding tool to facilitate bug finding and fixing and to verify static analysis alarms to ensure program correctness. The first set of tasks was designed to investigate developer bug finding and bug fixing patterns in software specifications written in a user-friendly specification language, Alloy [46]. This study was conducted online. The next set of tasks was designed to explore how developers inspect static analysis alarms to ensure program correctness and find defects. These tasks were placed on real-world open source C programs to test the defect detection skills of developers in a realistic situation. A questionnaire-based online study and an eye tracking study with these sets of tasks were conducted. For each mentioned study, two cognitive tasks [31, 32] are also included to explore the relationship of specific cognitive abilities such as working memory and spatial cognition with programming proficiency and certain skills of programmers [44].

The first study presented in Chapter 2 was on the comprehension of the Alloy specification language, which is a declarative lightweight formal specification language that uses first-order logic, sets, and relations to describe structures of systems and checks their important properties. Its companion tool, the Alloy Analyzer, is a model finder that finds instances of a specified model or checks properties related to a model. Alloy is used to formally specify software systems and check specification properties on these systems to prevent software faults at the design level. It was a suitable choice for conducting an experiment to explore program comprehension in declarative languages since it is easier to read and understand compared to other formal languages, and the Analyzer provides automated model checking and significant help to the user to find issues with the model [47]. Thirty participants with varying experience levels working with Alloy were recruited and were presented with a set of Alloy models that included bug finding and bug fixing tasks for them to work on. The steps that the participants took while solving the tasks were analyzed by capturing logs of the model when an Analyzer action was performed. The effect of experience level was examined by comparing the problem solving patterns of novices and non-novices in Alloy. The relationship between proficiency in solving Alloy tasks and working memory and spatial cognition skills was also explored in the study.

Chapter 3 presents the online study on assessing static alarms repositioning in the C language, and Chapter 4 reports on the eye tracking study conducted with the same goal. Developers spend a significant amount of their time working on fixing bugs [48], which requires thorough program comprehension. Even though using static analysis tools can help with catching a large percentage of defects, these tools are widely underused due to a large number of false-positive warnings and the high manual inspection load on developers [15]. To reduce the number of alarms and the manual effort of inspecting them, automatic repositioning of static analysis alarms has been

proposed [49]. The reduction in manual effort is expected due to repositioning the alarms closer to their cause points. The empirical study of the realistic effects of this proposed method was conducted in two separate studies. In the online study, 249 developers worked on software tasks that presented the type of alarms a static analysis tool would generate, helping us to empirically determine whether repositioning helps the users spend less time and effort finding the most useful static analysis alarms. In the eye tracking study, 27 students and professional developers were recruited to work on the same tasks on computers that were equipped with an eye tracker. The community eye-tracking infrastructure iTrace [50] was used to determine the source code elements developers were examining while solving the tasks, giving us a more detailed insight into the problem solving aspect of the tasks compared to the online study. Both studies also included two cognitive tasks to test working memory and spatial cognition in order to explore the relationship between these cognitive skills and the comprehension skills of the developers.

1.1 Research Contributions

The research contributions of this dissertation are as follows:

- Three empirical studies are conducted to gauge the comprehension of developers on the Alloy specification language (online survey) and verification of static alarms tasks in the context of bug fixing (online survey and eye tracking).
- First eye tracking study conducted on C code to verify static analysis alarms.
- Insight into the comprehension patterns of developers while working on various bug detection tasks.
- De-identified datasets and replication packages of developer data from the static analysis alarm verification study and Alloy study.

- Study of cognitive measures from developers that were correlated with the performance of software tasks.
- Suggestions and guidelines to improve the languages and tools designed to assist developers in defect detection and correction.

Chapter 2

The Effect of Experience on Bug Fixing Tasks in Alloy Models

2.1 Introduction

The Alloy formal specification language [46] aids in constructing models for structures of systems in the design phase and is useful for checking whether specific properties of systems hold. Its back-end tool, the Alloy Analyzer [51] performs automated analysis on models, checks assertions and generates counterexamples to those assertions if they do not hold. It also generates instances of models that can be used to show specific desired scenarios. Alloy has been used in a wide range of applications, such as test case generation [52], security analysis of Android [53, 54] and IoT devices [55], and verification of critical properties of real systems [56, 57]. Traditionally, working with formal specification languages has required in-depth mathematical knowledge due to their complexity, and the learning curve is very steep for non-mathematicians. In dealing with formal methods and designing formal specification languages, the most important factors have been soundness and correctness, and factors like readability and comprehension were often overlooked [16, 58].

Alloy's design seeks to alleviate this problem, with its easy-to-understand syntax and its use of familiar mathematical concepts such as sets. However, there is very little research on the usability and comprehension of Alloy as a language from the

user perspective and how it can be best taught to novices. Krishnamurthi et al. make a case for paying more attention to human factors in formal methods and state that performing more user focused research can be beneficial for building better tools and learning how to encourage more people to learn formal methods [16]. There has been some work on how students use Alloy Analyzer in different contexts [59, 60]. However, none of this work is focused on the comprehension of the Alloy language. This is important to study because even though the Alloy Analyzer is helpful, its use does not indicate comprehension of the actual Alloy model specifications. Moreover, almost all of the prior work is done with non-novices. There is a clear existing gap of human factor studies in the literature to understand how expertise plays a role in the comprehension of Alloy specifications. It has been shown that including varied expertise in program comprehension studies can give interesting insights on how developers think about problem solving [61].

The chapter presents an empirical study that seeks to understand how experience plays a role in the comprehension of the Alloy language while fixing bugs. Both novices and non-novices participated in the experiment. We present participants with natural language specifications for problems and their corresponding Alloy models, which included buggy statements. The participants were tasked with fixing the problems within the models so they matched their specifications. To the best of our knowledge, this is the first study to assess the impact of experience on the ability to fix syntactic and semantic bugs in Alloy models to match specifications. Research has shown that cognitive skills such as spatial recognition ability and working memory capacity are correlated with mathematical ability [36, 40]. In software engineering studies, a moderate correlation between working memory capacity and fixing bugs was found by Baum et al. [44] whereas Sharafi et al. found that spatial ability and data structure manipulation are correlated [45]. We wanted to test if there exists such a correlation

in fixing bugs in Alloy models. In order to do this, we performed two sets of cognitive tests (mental rotation [31] and operation span [32]) to explore the correlation between memory capacity and solving Alloy problems.

The research objective is to understand how novices and non-novices comprehend Alloy models while trying to find and fix bugs. The contributions of this study are as follows:

- An empirical study that explores comprehension of the Alloy specification language in different categories of bug fixing tasks.
- Comprehension pattern differences between novice and non-novices in Alloy. We are unaware of any previous study that has examined these differences.
- A detailed set of interactions on how Alloy Analyzer is used while fixing bugs.
- Cognitive tests to investigate the relationship between working memory capacity and spatial cognition ability with software modeling, also not studied before.
- A complete replication package for verifiability and replication purposes.

Results indicate that non-novices find and fix Alloy bugs with significantly higher accuracy (54% more accurate on average, Mann-Whitney U $p = < 0.001$), and on average, they complete all the tasks 32 minutes faster than novices. These results show that even a few months of familiarity and working with the Alloy language can make a big difference in levels of comprehension. Additionally, we found that the number of Alloy Analyzer actions and model edits correlated for all tasks, and these variables can predict the accuracy score of novices. These results imply that utilizing the analyzer can be a great help to novices. We found that both novices and non-novices make incremental changes before running the model again, to see if they can see a correct instance or fix the issues that create counterexamples. On average, novices make more changes to the Alloy models to get to the correct specification (average of 12

more edits for novices compared to non-novices over all the tasks). We found that non-novices had better scores in spatial recognition, which can be an indicator of the importance of this cognitive skill on understanding Alloy's underlying mathematical concepts.

2.2 Research Questions

We seek to address the following the research questions:

RQ1. What is the difference in bug fixing accuracy in Alloy models between novices and non-novices?

RQ2. What is the difference in bug fixing speed in Alloy models between novices and non-novices?

RQ3. What trends do we observe in behavior between novices and non-novices while fixing bugs?

RQ4. How do working memory and spatial cognition ability relate to bug fixing correctness?

The results from RQ1 would help us explore the differences between the novices and non-novices' understanding of the Alloy tasks and their accuracy in fixing bugs. RQ2 would help us quantify the time spent on fixing the bugs in each model between novices and non-novices. Both of these questions can help us understand how prior exposure to Alloy makes a difference in problem solving while fixing bugs. RQ3 helps us understand in more detail the patterns of bug finding in both groups via recorded snapshots of the specification every time a participant performed an action using the Analyzer. It also gives us insight into how each group fixed different types of bugs in the Alloy specifications. Finally, RQ4 explores the relationship between performance

on the Alloy tasks and two different cognitive skills that are related to mathematical and programming abilities, spatial cognition ability and working memory capacity.

2.3 Related Work

Besides empirical studies done on usability and comprehension in Alloy, this section also extends to studies on teaching and learning Alloy since there are not many studies assessing Alloy comprehension.

2.3.1 Studies on Usability and Comprehension in Alloy

The two studies most related to ours are Li et al. [60] and Danas et al. [59], who performed empirical studies on the Alloy language on novices. Li et al. [60] explored how the Alloy tool is used in practice by beginners, by logging some of the user interaction with the Alloy tool when students were building Alloy models. The students are asked to build Alloy models, which can be an indicator of their language comprehension. They found that users perform consecutive analyses after making small and incremental changes to the models and the users' interactions with the analyzer are similar to continuous compilation. In our study, we have focused on exploring comprehension by using bug fixing tasks. One of the most common tasks software developers perform is bug fixing, and understanding how they would find faults in an Alloy model and how they would correct those faults can give us more detailed information about their comprehension of the Alloy language. Unlike their study, we focus on both novice and non-novices and their problem solving skills. We also compare our findings from the logs acquired from our participants' problem solving process with Alloy.

Danas et al. [59] performed studies on both students and Mechanical Turk participants with the aim of exploring how different types of outputs of the Alloy Analyzer model finder are used in practice. They explored principled forms of output (such as minimal and maximal forms), provenance, and unsatisfiable cores. Their goal was to see how the different types of outputs help in understanding and debugging Alloy models, and they found that the output forms were not effective and helpful, and could be misleading in some cases.

Our work is complementary to both of these studies. We focus on how participants find and fix different kinds of bugs (syntactic and semantic) on Alloy specifications, with regards to the natural language specification. This can be an indicator of how comfortable the participants are in understanding the Alloy syntax and language, and how they can work with the Alloy Analyzer. Our study is the first to include a roughly equal number of novice and non-novice participants (Novice: N= 17, Non-novice: N = 13) in an empirical study on Alloy. The comprehension model and patterns of using the Alloy Analyzer can be observed in both of the groups, and can be compared to some of the findings of [60]. To our knowledge, this is also the first study to explore the relationship between specific cognitive abilities and comprehension of a lightweight formal language such as Alloy, and exploring these relationships can give us insight on what skills are more indicative of better comprehension of specification languages.

2.3.2 Studies on Teaching Alloy

Boyatt et al. [62] detail their experience on teaching Alloy as the main language for their Formal Methods module for undergraduate students. They mention that the automated feedback of the Analyzer and the readability of the language were factors in choosing Alloy for their teaching purposes. They explore the benefits of using an automated tool in teaching lightweight formal methods. They believe that using this

approach encourages students to use formal methods, as learning the language and working with the analyzer does not require learning difficult mathematical concepts, and the feedback it provides encourages the students to explore it further. They also state that they have seen the effects of using Alloy in improving abstract thinking and modeling and believe that its immediate feedback helps encourage students to explore modeling and abstraction. This work provides insight into how using Alloy can help with abstract thinking and problem solving, but it mostly focuses on the overall benefits of this teaching approach and does not study the problem solving methods and comprehension of students while using Alloy. Similarly, Noble et al. [63] introduce their method of introducing Alloy in one of their software engineering courses and how they leveraged the language to teach modeling, analysis, and verification of properties. They also focused on pedagogy, but the class evaluations show that students enjoyed their class and working with Alloy. However, despite reporting students' interests and grades in the class, their focus is not on comprehension and understanding Alloy.

Simonot et al. [64] explored the connection of mathematics and software engineering by using Alloy as a lightweight formal methods tool, and designed a course in which students work on Alloy problems in the lab after they learn different math concepts in class (such as logic, sets, functions, and relations). The paper describes the curriculum with the new approach and provides findings on educational aspects of using Alloy for teaching math; the paper also explores how to connect math and software engineering through using computer aided tools such as Alloy. However, this work does not focus on how students understand and fix Alloy models and problems either.

Tarkan et al. [65] describe teaching both Z and Alloy concepts through examples to 8 students, and since Alloy is based on the Z specification language, they explore how students can transfer the concepts they have learned from one language to the other. This study assesses the students' understanding of the language. After showing a

tutorial that involves different Z and Alloy models and includes many small questions about the complete models, students are asked four quiz questions, which include the completion of one line of Alloy code and the correction of a small model. Everyone who participated in this study was a novice and the researchers evaluated the accuracy of the survey questions but did not explore comprehension patterns. They found that the tutorials were effective in teaching both languages, and the novices were able to correctly answer questions about the models after a few hours of study.

Brown et al. [66] had students work on logic and relational algebra problems using Alloy, and taught these concepts using small problems in lab environment. The tasks in this paper were exclusively small logic and math based tasks, and only novices were asked to work on the problems. In comparison, our tasks are diverse (using a design specification, a data structure specification, and a puzzle specification) and we want to see the differences between novice and non-novice learning patterns as well. The purpose of their study is exploring how Alloy can be used for discrete math teaching purposes in a classroom environment. Their study design compares student performance in discrete math classes in two separate groups (one group learning with Alloy and the other without learning Alloy), but they do not explore Alloy comprehension in students either.

Macedo et al. [67] presents Alloy4Fun, a web-based application for sharing and working on Alloy models, which provides automatic assessment of simple challenges. They use this application while teaching an Alloy course in their university. They taught Alloy to 17 students for 5 weeks, and each lecture was followed by a lab session. They had a limited version of the Alloy Analyzer that only showed the graph visualization with no opportunity to choose different solvers and use other functionalities. They gave students different challenges on different problems and explored the data on students' interaction with the web application and quantitative

analysis on the feedback students received from the tool during their semester of learning Alloy and working on Alloy problems. They found that students had the most problems with solving the challenges that required more than ten logic or relational operators, manipulating ternary relations, transitive closure over expressions, and reasoning about total orders.

In contrast to the above studies, the study we present is different in that it focuses on assessing how developers fix syntactic and semantic bugs in Alloy models so that they conform to natural language specifications. We also monitor and compare behavior interactions of novices and non-novices with the Alloy models and a fully functional Alloy Analyzer that allows them to choose solvers during bug fixing.

2.4 Experimental Design

In this section, we describe the experimental design of the study, including participants, tasks, study instrumentation, and measures. A complete replication package[1] is available for download. The informed consent form and the tasks for the Alloy study are available in Appendix A.

2.4.1 Experiment Overview

The goal of our controlled experiment [68,69] is to evaluate and explore comprehension of the Alloy specification language for the purpose of evaluating bug fixing behavior from the point of view of novices and non-novices in the presence of syntactic and semantic bugs. We designed two different task types to help us study Alloy comprehension in our participants: Alloy tasks and cognitive tasks. The Alloy tasks are designed for the participants to locate and fix syntactic and semantic bugs in the Alloy models.

[1] `https://osf.io/5p6e4/?view_only=861a8de10ce249fa92eda4b9ac7776cd`

Table 2.1: Experiment Overview

Goal	Study Alloy specification language in context of bug fixing
Independent Variable	Experience (novice and non-novice)
Task Types	Alloy Tasks: Syntactic Alloy Error, Semantic Alloy Bug
Dependent Variables	Accuracy, Speed, Usage of Analyzer (Number of Analyzer Actions) Number of Edits
Secondary Factors	Operation Span Task Score, Mental Rotation Task Score

The cognitive tasks measure working memory capacity and spatial recognition ability (RQ4) to determine if these tasks play a role in bug fixing performance. We measure comprehension using accuracy, speed, number of Analyzer actions, and number of edits. An overview of the experiment is shown in Table 2.1. The study was approved by the university's institutional review board.

2.4.2 Participants and Experience

We recruited 30 participants from different universities and institutions worldwide. Each potential participant was sent an email inviting them to participate in study, and if they accepted the invitation, they were assigned an ID to enter on the questionnaires of the study and were sent the study package via email along with a consent form. When the participants submitted the study, they were compensated with a $10 Amazon e-gift card.

Our participants had different levels of expertise in Alloy, ranging from beginners who were just learning the language, to experts who have been working with the language for years. Participants were recruited through emails to class mailing lists,

posts on Alloy messaging boards, and through professional contacts. They were asked to fill out a demographic questionnaire before the start of the study, which asked them about their age, gender, affiliation, degree, their native language, and proficiency in English. There were 19 male participants and 11 female participants. Eleven participants were between 21-25, nine participants were between 26-30, seven participants were between 31-35, two were 36-40, and one was over 40 years old. 29 participants were either pursuing or had Computer Science, Computer Engineering, or Software Engineering degrees, and one was pursuing a Industrial and Labor Relations degree. Three participants were either pursuing a bachelor's degree or held one. The rest of the participants were either pursuing a graduate degree or held Master's or Doctorate degrees. Out of the 30 participants, twelve of them stated that English is their first language, and the self-reported assessment of fluency in English language of the remaining is as follows. Two participants ranked their fluency 3 out of 5, eleven participants ranked their fluency 4 out of 5, and seven participants ranked their fluency 5 out of 5.

We asked the participants to self-report their experience level, as it has been established [70] that self estimation is a reliable measurement for programming experience. Participants completed a post-questionnaire (after they completed the study to avoid any imposter syndrome bias) that asked them to rate their programming skills, design skills, knowledge in set theory, first order logic, and object oriented languages skills. They were also asked to rate their comprehension level of Alloy syntax and their level of comfort using the Alloy Analyzer. All these questions were answered by making a choice on a scale of 1 to 5, from Poor to Excellent. We asked the participants how many years of experience they have in working with programming languages, specification languages, and Alloy. And finally, we provided a list of formal specification languages and tools and asked the participants to indicate if they

have experience working with them. Based on the post-questionnaire, we defined non-novices in Alloy as having more than 1 year of experience, or having less than one year of experience but having familiarity with the language and rating their comfort level in understanding Alloy syntax higher or equal to 3 out of 5. With this criteria, our novice group consisted of N = 17 and non-novices of N = 13.

2.4.3 Tasks

The first category of tasks are the cognitive tasks. The two cognitive tasks were the Operation Span Task [32, 71] and the 3D Mental Rotation Task [31], which measure working memory and spatial recognition ability respectively. Prior research [44, 45] has shown a correlation between cognitive tasks and software comprehension tasks. We aimed to explore whether these correlations exist for bug fixing tasks in Alloy.

We used a Python version of the Operation Span task [72] for our study. This task shows a number of letters to the participant, with a distractor math task between each letter, and asks the participant to recall all the letters they have seen in order. The final calculated score (partial-credit unit score) is between 0 and 1, with a score of 1 indicating that the participant recalled all the letters correctly. Each participant completed four practice trials and 12 task trials.

We implemented a Java desktop application for the 3D Mental Rotation task. The task shows an image of a 3D object to the participant and asks the participant to choose the correct rotations of the 3D object from four different images presented to them. For this task, each participant completed five practice trials and 20 task trials. Since the participants had to choose two correct rotations of the 3D object, we gave them 1 point for each correct choice they made. This totaled the overall score for this task to 40.

Table 2.2: Syntactic bugs in Alloy models
(Buggy statements are bold and in the color blue)

Model Name	Syntactic Bugs	
	Original Specification	Altered Specification
grade.als	PolicyAllowsGrading**[s,a]**	PolicyAllowsGrading**[s]**
balancedBST.als	Node = BinaryTree.root**.***(left + right)	Node = BinaryTree.root***.**(left + right)
farmer.als	s0.near = Object **&&** no s0.far	s0.near = Object **&** no s0.far

Table 2.3: Semantic bugs in Alloy models' facts or signature constraints

Model	Semantic Bugs	
	Change a fact or constraint	
	Original Specification	Altered Specification
grade.als	associated_with: **one** Class	associated_with: **set** Class
balancedBST.als	**lone** n.∼(left + right)	**some** n.∼(left + right)
farmer.als	Farmer in s.near => crossRiver[s.near, s'.near, s.far, s'.far] else crossRiver**[s.far, s'.far, s.near, s'.near]**	Farmer in s.near => crossRiver[s.near, s'.near, s.far, s'.far] else crossRiver**[s'.far, s.far, s'.near, s.near]**

Table 2.4: Semantic bugs in Alloy models' predicates

Model	Semantic Bugs	
	Fix a predicate	
	Original Specification	Altered Specification
grade.als	s **!in** a.assigned_to	s **in** a.assigned_to
balancedBST.als	**all nl: n.left.*(left + right) | nl.elem** <n.elem **all nr: n.right.*(left + right) | nr.elem** >n.elem	**some n.left =>n.left.elem** <n.elem **some n.right =>n.right.elem** >n.elem
	(HasAtMostOneChild[n1] && HasAtMostOneChild[n2]) => (let diff = minus[Depth[n1], Depth[n2]] — -1 <= diff **&&** diff <= 1)	(HasAtMostOneChild[n1] && HasAtMostOneChild[n2]) => (let diff = minus[Depth[n1], Depth[n2]] — -1 <= diff **||** diff <= 1)
farmer.als	(one item : from - Farmer | { from' = from - Farmer - item - **from'.eats** to' = to + Farmer + item })	(one item : from - Farmer | { from' = from - Farmer - item to' = to **- to.eats** + Farmer + item })

Before diving into the Alloy tasks, we introduce some important constructs of the Alloy specification language. Each model consists of *signatures*, which define basic sets and relations and provides the vocabulary of the model, and formulas that introduce constraints on those relations. *Facts* are statements that are always assumed to be

true for the model, and the analyzer discards of the instances where any fact statement is violated. *Predicates* can accept parameters and consist of constraints that can be evaluated to be either true or false. Finally, *Functions* are expressions that return a relation and can accept input parameters as well. The Alloy Analyzer can find valid instances of a model and show counterexamples to assertions about the model. For more details, we direct the reader to [46] for more details on the constructs of the Alloy language and the Alloy Analyzer tool.

We chose three Alloy models from the GitHub repository by Wang et al. [73], located at [74]. The three models we chose are *grade.als*, *balancedBST.als*, *farmer.als*. These models were chosen because they described three different types of problems (non-technical, data structure, and logic puzzle), and their difficulty levels were different. The Grade model describes a gradebook designed to include constraints about the graders and classes and is a fairly simple concept for everyone to understand. The BST model specifies a balanced binary search tree, which is a data structure people with computer science backgrounds are familiar with. The Farmer model seeks to solve the classic River Crossing Puzzle [75] and requires logical problem solving abilities. Due to their various levels of complexity, we label these models as easy, medium, and difficult models, respectively. We also provided the natural language specification, explaining what problems these Alloy specifications are modeling. The participant was instructed to change the model in a way that it would adhere to the natural language specification. We used some of the bugs in ARepair's [73] buggy models but introduced some other other bugs to fit our task types: syntactic bugs and semantic bugs. In introducing syntactic bugs, a line in an Alloy model was changed (Table 2.2). These bugs elicit errors from the Alloy Analyzer that would help the user in detecting and fixing them, and fixing the bugs requires a level of understanding of Alloy syntax. We also introduced semantic bugs into the models. We changed either

the facts of the model or constraints within a signature to modify the constraints of the model, as shown in Table 2.3. And finally, we also changed some predicates in the models to change their meanings (Table 2.4). The semantic changes resulted in the Alloy Analyzer showing incorrect instances or counterexamples to assertions, and the participants were expected to find and correct these bugs.

Since the models had different levels of difficulty, we permuted the order of the tasks to control for order effects (3! Alloy tasks $\times$ 2! Cognitive tasks), ending up with 12 different variations of the study. The participants were always asked to do the cognitive tasks first and then work on the Alloy tasks, but the order of the tasks was randomized within each category (Alloy or Cognitive). Table 2.5 shows an example task order for a participant. Each participant had to complete two cognitive tasks and fix 10 Alloy bugs in total. The participants were not aware of how many Alloy bugs there were. The only instructions given to them were to make sure the Alloy model conforms to the natural language specification.

2.4.4 Dependent Variables

We modified the Alloy Analyzer and instructed the participants to use the modified version of Alloy while working on the tasks. The modified Analyzer logged snapshots of the open Alloy specification file every time the user executed a command. The logged user actions were as follows. We also logged the timestamp of the actions performed by the user.

- *Execute:* Runs the most recent or the first written command if no command has been executed so far. The commands can be either "assert" for generating counterexamples to an assertion, or "run" for generating instances of a predicate.

The command "run" can be combined with "show" to show an instance of the model.

- *Show Instance:* Displays the most recent instance or counterexample.
- *Show MetaModel:* Displays a *meta model* of the currently open Alloy specification [46], which shows the relationships between different elements (e.g., signatures) of the specification as an object model.

Based on the above logs, we derive the following dependent variables.

- *Accuracy:* Accuracy is calculated by assigning 1 point to each correct bug fix, and half a point for localizing the bug. There were cases where a participant would localize the line of the bug but not be able to fix it correctly. The maximum score a participant could receive from all the tasks was 10 points. Tasks for models Grade and Farmer had a maximum of 3 points, and tasks for the Balanced BST model had the maximum of 4 points.
- *Speed:* Speed is calculated by looking at the self-reported start time and finish time of the tasks. The particiapnts questionnaire had locations to enter the start and end wall clock time.
- *Number of Analyzer Actions:* We used the information from the logs to calculate how many times they performed an action (Execute, Show Instance, etc) on each model to see instances or check assertions.
- *Number of Edits:* We used the information from the logs to calculate the number of times the participants edited each model.

2.4.5 Study Instrumentation

We sent emails to potential participants inviting them to participate in the study. Once they accepted the invitation, another email was sent including the link to the

study package. The participants did the study remotely in the location of their choice. The study package included the two categories of tasks (cognitive and Alloy tasks), a modified version of the Alloy Analyzer, a tutorial on Alloy for participants to either learn or remember the language syntax and structure, a sample task on Alloy with the correct answers, and a ReadMe file detailing the steps to do the study. We also asked the participants to fill out pre and post questionnaires to gather demographic and self-reported experience data. Participants were instructed to read through a ReadMe file that walked them through the steps of the study. They were also required to make sure that they had Python 2.7 and Java installed on their machines to be able to run the cognitive tasks. The participants were asked to record the start and finish time of working on each Alloy model, and state their level of confidence in their answer and provide any additional comments if needed. The participants were also asked to do the study in one sitting and without interruption, and then submit the study package including all their changed files and the generated logs back to the researchers. Note that the study was not conducted via a web browser since we wanted to take full advantage of all the functionalities of the Alloy Analyzer, which is not available for the web.

2.5 Experimental Results

First, we discuss the pre-processing steps followed by results and hypothses listed for each RQ.

2.5.1 Pre-processing

We created a master file that included the cognitive task and Alloy task data for each participant. For the operation span task, we gathered the automatically graded scores

Table 2.5: Example task order for a participant

Task Type	Group	Task
Cognitive	-	Operation Span Task
		3D Mental Rotation Task
Alloy	Easy Specification: Gradebook (grade.als)	Find and Fix Syntactic Error
		Find and Fix Semantic Error (x2)
	Medium Specification: Balanced Binary Search Tree (balancedBST.als)	Find and Fix Syntactic Error
		Find and Fix Semantic Error (x3)
	Difficult Specification: Farmer Puzzle (farmer.als)	Find and Fix Syntactic Error
		Find and Fix Semantic Error (x2)

of the working memory tasks from the generated files, and for the mental rotation task a python script was written to grade the result files. The automated nature of grading these two tasks eliminates the possibility of errors in grading. For the Alloy task data, a Python script was written to show the differences between the Alloy files, by creating an HTML file that highlighted the differences. Seeing the highlighted differences in a web browser facilitated the grading of the tasks. We used the highlighted differences to grade each submission, by looking at the submitted version of the model next to the original version that included the bugs. One of the authors ran each of the submissions to make sure they passed all the checks. We did not auto grade Alloy tasks via a script since there were multiple ways of fixing a bug in some cases. The

manual nature of running all the submissions show that the bug was either fixed or not fixed leaving no subjective nature to the grading. When looking at the differences between the original and submitted files, if a change was made to a buggy line, we consider this as bug localization. The change might not necessarily have fixed the bug correctly. Some participants also commented the lines with "bug detected". Refer to Section 2.4.4 for scoring criteria.

Aside from the submitted models from each participant, a number of files were generated by the Analyzer if the participant used it to see instances of the model or counterexamples to an assertion. These files include the snapshot of the Alloy model at the time when the user performed an action, the type of the action performed (execute, show instance, show metamodel), and the timestamp of when the snapshot was saved. We refer to these snapshots as "logs". Due to technical difficulties, we were unable to process 3 log files. We wrote a Python script to extract the action sequence and time spent between each action from the log files, and we used the difference finder to go through all the logs submitted by the user for each model to show the differences between the snapshots after each action was performed. For every set of logs that we had (for different participants and models) we generated an HTML file showing all the changes and highlighting the steps the participants took to localize and fix the Alloy bugs. From these HTML files, we acquired information about the number of edits. All these calculated measures for each participant were added to the master file. We used JASP [76], a free program for statistical analysis to perform statistical tests on our data.

2.5.2 RQ1 Results: Accuracy

Research question 1 asks about the accuracy of novices and non-novices when fixing Alloy bugs. The null and alternate hypothesis are as follows.

Table 2.6: Descriptive Statistics for Accuracy Across the Tasks and Models

	AccuracyScore		**AccuracySyntactic**		**AccuracySemantic**		**AccuracyGrade**		**AccuracyBST**		**AccuracyFarmer**	
	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice
Valid	17	13	17	13	17	13	17	13	17	13	17	13
Mean	5.588	8.615	2.382	2.923	3.206	5.692	2.147	2.731	1.735	3.308	1.706	2.577
Std. Deviation	2.386	1.024	0.740	0.188	1.937	0.925	0.880	0.599	0.970	0.855	0.902	0.277

Note. AccuracyScore is the overall score the participants received for all the tasks in all the models. AccuracySyntactic and AccuracySemantic are the scores received in semantic and syntactic task types, respectively. AccuracyGrade, AccuracyBST, and AccuracyFarmer are the scores received from solving the tasks in each model.

AH_0 Having experience (non-novices) working with the Alloy specification language does not have an effect on the accuracy of solving the tasks.

AH_A Having experience (non-novices) working with the Alloy specification language has an effect on the accuracy of solving the tasks.

We used the participants' accuracy score on Alloy tasks as a measure of program comprehension. One of the authors graded all the submissions and compared the final submitted version of each Alloy specification file with the correct specification file. For each syntactic or semantic bug, if the participant changed the buggy line, they received score of 0.5 for that bug due to successfully localizing the bug. If the participant changed the buggy line and corrected the bug, they received a score of 1 for that task. There were overall 3 syntactic bugs and 7 semantic bugs across the three Alloy models, and the maximum score a participant could receive was 10.

Table 2.6 presents the descriptive statistics. Overall, non-novices performed better on the tasks and the average of accuracy score for non-novices ($M = 8.615 \pm 1.024$, $N = 13$) is 54.17% higher than the average of accuracy score for novices ($M = 5.588 \pm 2.386$, $N = 17$). We observe the same pattern in individual models and on different types of tasks as well. Figure 2.1 shows the box plots of the overall accuracy score in both groups. We can see that the scores are widely dispersed in the novice group, ranging from 1.5 to 8.5, whereas the non-novice group's scores range from 6 to 9.5, with the minimum score of 6 being an outlier in this group. We were also interested in the

differences between the scores of all participants in syntactic and semantic task types. We observe that the participants performed better in syntactic bug fixing tasks in general, with the average of M = 2.61 ± 0.625 (maximum score of 3) compared to semantic tasks, with the average score of M = 4.283 ± 1.99 (maximum score of 7).

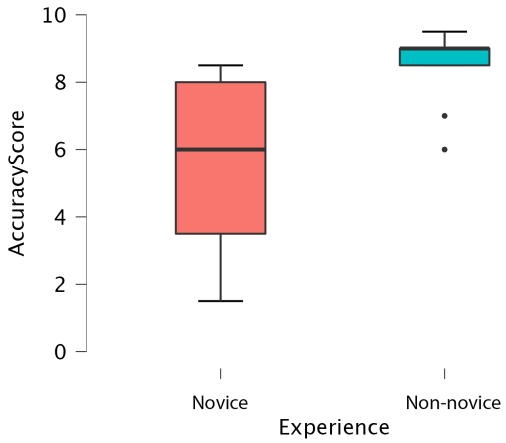


Figure 2.1: Box plot of overall accuracy across Alloy tasks.

To investigate whether the differences between these two groups are statistically significant, we first performed the Shapiro-Wilk test to test the assumption of normality, and realized that the data was not normal in all the groups. For the non-normal data, we chose to perform Mann–Whitney U test, a non-parametric test to compare the novice and non-novice groups in all the different accuracy scores. Based on the Mann-Whitney U results in Table 2.7, there are significant differences between the total accuracy scores of two groups ($p < .001$), accuracy in Grade specification tasks ($p = .032$), accuracy in BST specification task ($p < .001$), and accuracy in Farmer task ($p = .005$). We also observed that significant difference can be seen between novice and non-novice groups in fixing syntactic ($p = 0.006$) and semantic ($p < .001$) bugs as well. The significant differences between the two groups, gives us evidence to reject the null hypothesis which means that experience makes a difference in solving Alloy tasks correctly.

RQ1 Finding: Non-novices performed significantly better in different task types and overall compared to novices. Participants received higher scores on syntactic tasks compared to semantic tasks.

Table 2.7: Mann-Whitney U Test Results for Accuracy in Two Groups

	W	p	Rank-Biserial Correlation
AccuracyScore	18.500	$< .001$	-0.833
AccuracyGrade	63.000	0.032	-0.430
AccuracyBST	23.500	$< .001$	-0.787
AccuracyFarmer	46.500	0.005	-0.579
Syntactic	52.000	0.006	-0.529
Semantic	18.000	$< .001$	-0.837

Note. For the Mann-Whitney test, effect size is given by the rank biserial correlation. W is The Mann-Whitney statistic (W-Value) is the sum of the ranks of the first sample

2.5.3 RQ2 Results: Speed

Research question 2 asks how novices and non-novices solve Alloy specification tasks in terms of speed. The null and alternate hypotheses are as follows.

TH_0 Having experience (non-novices) working with the Alloy specification language does not have an effect on the speed of solving the tasks.

TH_A Having experience (non-novices) working with the Alloy specification language has an effect on the speed of solving the tasks.

Table 2.8 shows the descriptive statistics for the speed for each specification and overall for both novice and non-novice groups. On average novices (Overall column, $\mathrm{M} = 81.235 \pm 66.39$, $\mathrm{N} = 17$) took 32 more minutes to finish the Alloy tasks compared to non-novices (Overall column, $\mathrm{M} = 49.077 \pm 15.78$, $\mathrm{N} = 13$). We can observe the same pattern in individual models as well. We ran the Shapiro-Wilk normality test

Table 2.8: Descriptive Statistics for Speed (in minutes) for Each Model and Overall

	Grade		BST		Farmer		Overall	
	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice
Valid	17	13	17	13	17	13	17	13
Mean	16.471	12.769	23.471	20.923	41.294	15.385	81.235	49.077
Std. Deviation	10.026	5.761	15.399	8.067	61.531	7.911	66.390	15.787

for this data. The distribution of the Grade model duration was the only normal distribution, where the t-test was used. For the rest of the tasks and the overall speed, we used Mann-Whitney U test to see if there are any significant differences between the groups. No statistically significant differences were found between the two groups, indicating not enough evidence to reject the null hypothesis (Grade t-test $p = 0.24$, BST Mann-Whitney U $p = 0.85$, Farmer Mann-Whitney U $p = 0.18$, Overall Mann-Whitney U $p = 0.28$). Additionally, a two-way ANOVA was performed to analyze the effect of difficulty and experience on task solving speed. It revealed that there was not a statistically significant interaction between the effects of difficulty and experience on task solving speed ($F(2, 84) = 1.580$, $p = 0.212$).

RQ2 Finding: We found that on average non-novices finished all the Alloy tasks 32 minutes faster than novices, but we did not find statistically significant differences in speed with the novice group.

2.5.4 RQ3 Results: Behavior Patterns

Research question 3 asks what kinds of behavior patterns are observed while solving the tasks. The null and alternate hypotheses are as follows.

PH_0 Having experience (non-novices) working with the Alloy specification language does not have an effect on the behavioral patterns of problem solving in Alloy.

Table 2.9: Descriptive Statistics For Number of Logs (Number of Performed Actions) and Edits

	GradeLogsNum		BSTLogsNum		FarmerLogsNum		TotalLogs	
	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice
Valid	16	10	15	11	16	11	16	11
Missing	1	3	2	2	1	2	1	2
Mean	9.375	8.800	26.333	15.818	24.750	7.909	58.813	31.727
Std. Deviation	8.921	6.909	26.351	15.823	32.460	5.467	51.763	23.946

	GradeNumberOfEdits		BSTNumberofEdits		FarmerNumberofEdits		NumberOfEdits	
	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice	Novice	Non-novice
Valid	16	10	15	11	16	11	16	11
Missing	1	3	2	2	1	2	1	2
Mean	6.500	9.200	17.600	16.091	20.875	7.273	43.875	31.727
Std. Deviation	5.633	7.525	18.212	16.802	30.785	6.389	39.673	24.816

PH_A Having experience (non-novices) working with the Alloy specification language has an effect on the behavioral patterns of problem solving in Alloy.

To address RQ3, we explore the data we gathered from the participant logs to find any specific patterns in the number of actions and number of edits in them. We believe that the number of actions and edits are useful quantitative measures about using the Analyzer and the work patterns of the participants. The Analyzer provides an interactive environment for the participants and gives us data on the number of actions (execute, show model, show metamodel) performed.

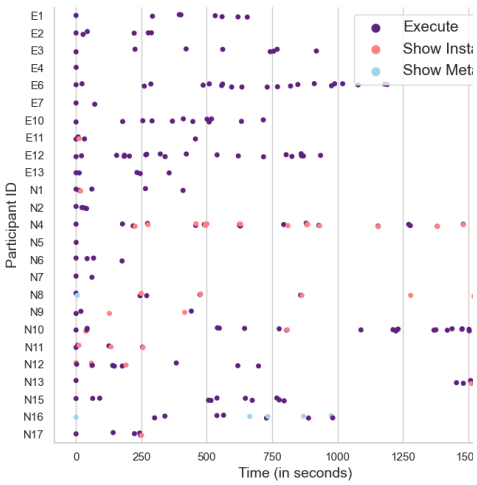
Table 2.9 shows the average number of actions and edits performed by the participants. The data shows that on average (rounded up to the nearest whole number), novices performed more actions ($M = 59 \pm 52$, $N = 16$) compared to non-novices ($M = 32 \pm 24$, $N = 11$). The same pattern can be observed in the Grade, Farmer and BST logs as well. Since the data was not normal (based on the Shapiro-Wilk test), we use the Mann-Whitney U test on the data, but did not see a statistically significant

difference between the number of actions each group performed, across each different model and overall (All p-values were greater than 0.05).

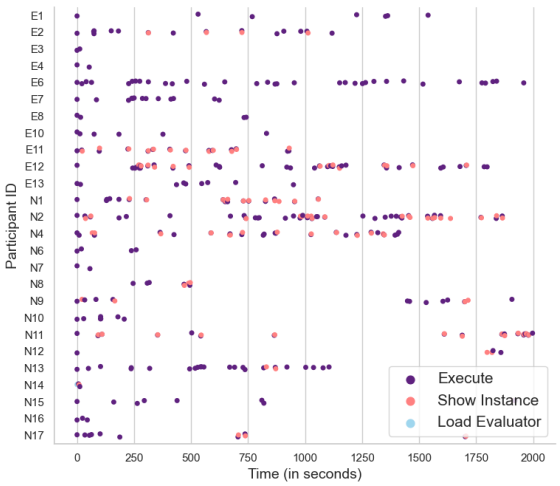
We were also interested in seeing what type of actions participants performed the most. Figure 2.2 show the categorical scatter plots of the sequence of actions performed by participants for each model. The Y axis shows the participant ID, in which the participants denoted with E1-E13 are non-novice participants and participants denoted with N1-N17 are novices. The X axis shows the time spent in seconds, starting from 0 to when the last log was recorded. The time was limited to 2000 seconds for Grade scatterplot, as 91.3% of participants completed the task in under 2000 seconds. The time was limited to 2500 seconds for the BST and Farmer tasks, in which 84% and 77.7% of the participants completed the tasks under that time. Limiting the time axis was done for better visibility of the data. We can see that as the tasks get more difficult (Difficulty: Grade $<$ BST $<$ Farmer), the number of actions performed by non-novices is reduced compared to the number of actions by novices. We can also see that "Execute" is the most popular action overall between both novice and non-novices. Interestingly, participants used "Show Instance" while working on the BST model the most, to see valid instances of the balanced binary search tree. We observe that most of actions are performed pretty close to the one before. We also noticed that changes were incremental: Most of the times, participants only changed one line and performed an action again.

Overall, we observe that novices make more edits than non-novices on average (Novice: M $= 43.87 \pm 39.67$, N $= 16$, Non-novice: M $= 31.72 \pm 24.81$, N $= 11$), but the Mann-Whitney U test did not show any significant difference between them ($p = 0.4$).

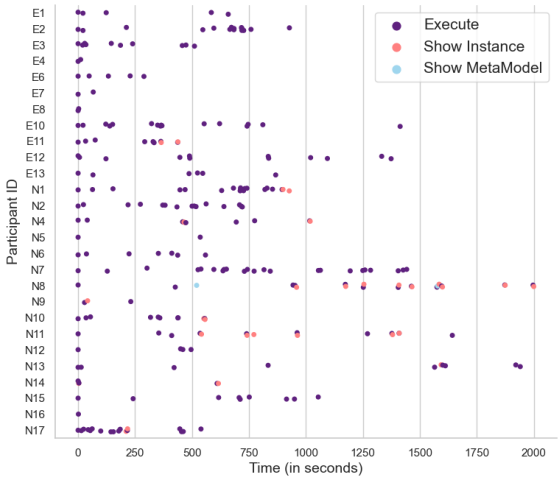
Additionally, we wanted to know whether performing more actions correlated with the number of edits in both of the groups. We looked at the correlation between the number of edits for each model and the number of actions on each model, and the



(a) Grade Action Sequence



(b) BST Action Sequence



(c) Farmer Action Sequence

Figure 2.2: Analyzer Action Sequences across two groups (E1-E13: Non-novices, N1-N17: Novices)

overall number of actions and edits. We found that for both groups, and for each individual model and overall, the number of actions correlated positively with the number of edits. This can indicate that seeing an instance of the model helped the participants with making edits. Finally, we ran the linear regression model with bug fixing accuracy as our dependent variable, and number of actions and edits as our independent variable in both novice and non-novice groups. The regression model was statistically significant in predicting the outcome variable, meaning that the number of actions and edits had a positive effect of novice group's overall score ($p = 0.046$, regression equation: Accuracy $= 4.192 - 0.07(NumberOfEdits) + 0.032(TotalLogs)$). We could not find this relationship in the non-novice group.

RQ3 Finding: We found that on average non-novices perform fewer actions and make fewer edits compared to novices. We observe that participants used the "Execute" action the most, and they made small and incremental edits before executing the commands again. Moreover, the number of actions and edits correlated with bug fixing accuracy for novices.

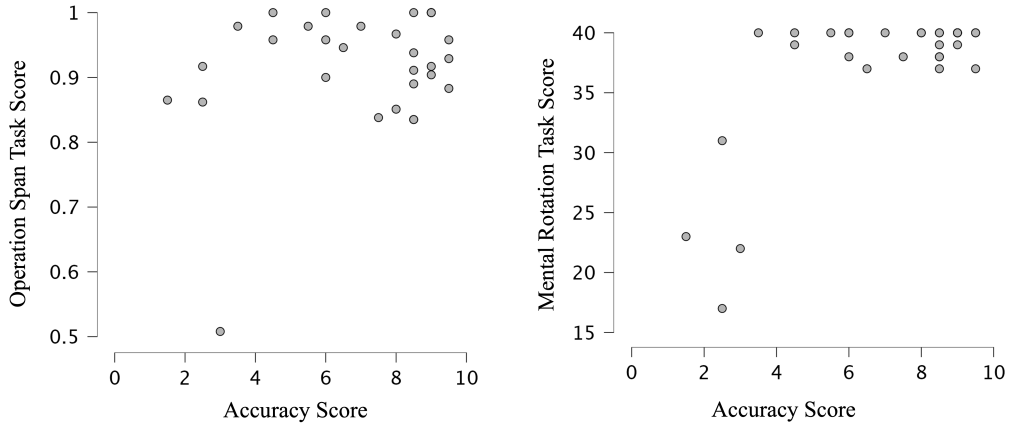
2.5.5 RQ4 Results: Working Memory and Spatial Cognition

Research question 4 asks how working memory and spatial cognition ability relate to task correctness. The null and alternate hypotheses are as follows.

$CogH_0$ There is no relationship between working memory and accuracy, and no relationship between mental rotation skills and accuracy.

$CogH_A$ There exists a relationship between working memory and accuracy and mental rotation skills and accuracy.

Figure 2.3 shows the scatter plots displaying the relationships between the operation span task scores and the overall task accuracy, and mental rotation scores and overall task accuracy. We ran Spearman's correlation to assess the relationship between



(a) Working Memory and Accuracy (b) Spatial Cognition and Accuracy

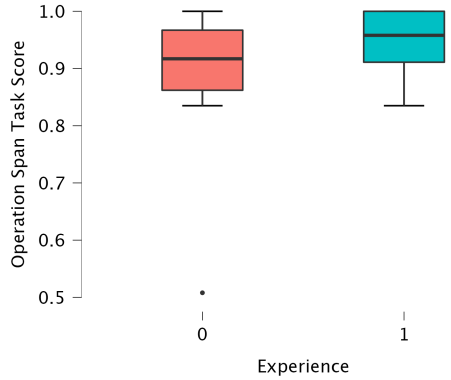
Figure 2.3: Scatterplots showing the relationship between cognitive tasks' scores and overall Alloy tasks accuracy score

the operation span task score and the overall accuracy score in Alloy tasks. The correlation was not statistically significant ($r_s = 0.103$, $p = 0.588$). Next, Spearman's correlation was run to assess the relationship between mental rotation task score and overall accuracy score in Alloy tasks. There was a positive correlation between the two variables, which was statistically significant ($r_s = 0.367$, $p = 0.046$).

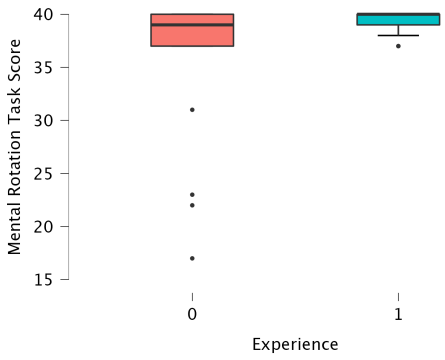
Furthermore, since we are aware of the meaningful difference in accuracy scores between the two novice and non-novice groups, we decided to explore if there is a meaningful difference between the mental rotation and operation span task scores of the novice and non-novice groups, which could indicate a relationship between higher accuracy score to any of the two cognitive scores. We do not claim that experience has an effect on cognitive tasks' scores. Figure 2.4 shows the boxplots of the cognitive task scores of the two groups. We are exploring the differences between the two groups as they received significantly different accuracy scores. Overall, participants (N = 30) earned an average score of M = 0.917 ± 0.095 in the operation span task (score range 0 - 1) and an average score of M = 37.1 ± 5.92 in the 3D mental rotation task (score range 0 - 40) (Scoring method explained in Section 2.4.3). Table 2.10 shows that the average

Table 2.10: Descriptive Statistics For Cognitive Tasks

	Working Memory Score		Mental Rotation Score	
	Novice	Non-novice	Novice	Non-novice
Valid	17	13	17	13
Mean	35.471	39.231	0.896	0.944
Std. Deviation	7.484	1.166	0.115	0.053



(a) Operation Span Score Between Two Groups



(b) Mental Rotation Score Between Two Groups

Figure 2.4: Boxplots showing the cognitive scores between two groups

working memory score is higher in the non-novices group (Novice: $M = 0.896 \pm 0.11$, $N = 17$, Non-novice: $M = 0.944 \pm 0.053$, $N = 13$). We performed a Mann-Whitney U test on the data, which did not inform of any significant differences between the two groups. We observe a difference in the average score of the mental rotation task between novices ($M = 35.47 \pm 7.48$, $N = 17$) and non-novices ($M = 39.23 \pm 1.16$, $N = 13$), but Mann-Whitney did not show a statistically significant difference between the two groups.

RQ4 Finding: We found that mental rotation task scores correlated positively with performance on Alloy tasks. On average, non-novices did better on the mental rotation and working memory tasks compared to novices, although this difference is not statistically significant.

2.6 Threats to Validity

Internal Validity: The Alloy community is a relatively small community. The models that were presented in this study can be considered educational Alloy models. It is possible that some of the non-novices who have had experience with Alloy might have seen these models before when they were learning or teaching Alloy. We injected 1 to 2 line bugs into the models ourselves. In order to have everyone at the same baseline to start, we asked all the participants to go through the Alloy tutorial we sent them. They were also presented with comments in the models that could help them find the bugs. We also asked the participants not to look online for answers, but since the study was remote we did not have any control over this factor. To mitigate this threat, we took every precaution to make sure the instructions given to the participants were clear.

External Validity: Finding Alloy users to take part in the study was definitely a challenge as this population is smaller than the general developer population. It was extremely difficult to recruit participants, and a number of students dropped out of the study because they stated that they do not understand Alloy and cannot solve the tasks. These students did not submit any tasks. Finding non-novices was also challenging because Alloy is mostly used in academic settings and finding experts who were willing to partake in the study was difficult. In spite of this, we secured 13 participants who knew and used Alloy before through extensive advertising and 17 who were willing to read the tutorial and learn the language before completing the study.

Construct Validity: All our dependent variables were chosen carefully to make sure they represent what we seek to measure. Even though we automated most of the log analysis, we manually validated the measures to mitigate any errors in calculation.

Conclusion Validity: The conclusion validity of the experiment is dependent on using the correct statistical tests to determine significance. 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.

2.7 Discussion and Implications

Our findings for RQ1 and RQ2 present clear differences between novices and non-novices in accuracy and speed of working on tasks. It implies that prior exposure to and experience with the language is important in completing Alloy tasks. Despite Alloy being more readable and easier to understand in comparison to other formal languages, it is still challenging for novices to work on Alloy tasks without much background on formal methods and the language itself. RQ3 results show that the novices rely more on the Analyzer to find issues with the model and make more edits to fix bugs. We also observe that overall and in each task, the number of actions and edits were correlated, indicating that the instance generated by the analyzer can help the participant on deciding about their edits. Furthermore, in the novice population we observe that accuracy is affected by the number of actions and edits, which implies that seeing the instances and interacting with the analyzer helps the novice participants in solving the tasks more accurately. RQ4 results show that while we could not see a statistically significant difference, comprehension of Alloy language is more related to the spatial cognition ability and not as much related to working memory capacity. We can rationalize this difference by pointing out that Alloy tasks are not memory intensive tasks, and they are more related to mathematical abilities of a person, which research shows is correlated with spatial recognition abilities [36].

The findings about the patterns in solving tasks, specifically the fact that the participants make incremental and small changes is consistent with the observations of Li et al. [60], who found that users perform consecutive actions on models that are only slightly different. The other study [59] is not comparable to ours because their tasks were different and dealt with which model outputs (minimality/maximality, UNSAT cores) people used the most. Based on the outcome of this study, we recommend the following improvements to the Alloy Analyzer as well as the areas of focus for teaching Alloy. Given that novice users struggled more with semantic tasks (while they performed relatively well on syntactic ones), teaching systematic methods for debugging an Alloy model to locate and fix semantic bugs more quickly (e.g., identifying an over-constraint that results in unsatisfiability of a model or a missing constraint that causes an assertion failure) is recommended; an extension to the analyzer that automates this debugging process would also be valuable. In addition, given that both novices and non-novices tend to work with the Alloy models in an incremental manner, tool enhancements that further facilitate this incremental process (e.g., automated compilation and execution of the model given a change; generating suggestions for which part of the model the user should inspect next) would also be helpful.

These findings have direct implications in the education and practice of Alloy modeling. Educators can make use of patterns we find in novices to better teach Alloy and help them avoid common mistakes while fixing bugs. In practice, one can choose developers in industry who are better at spatial skills to help with Alloy debugging. Formal specification languages such as Alloy are used for many safety critical applications [56, 57, 77]. With the amount of day to day activities that depend on software running safely and securely, it is important to study how developers interact with modeling software such that we can improve them to support the novice modelers by learning how the experts/non-novices behave. We still have a long way

to go in this area as is clearly evidenced from the literature. We strongly believe that more studies on formal specification languages are needed on many different types of tasks. The tasks used in this study are only the beginning to pave the way for more studies that can be conducted in this space. One way we can improve usability and tool support and adoption of software modeling tools such as Alloy is by learning via studies such as this one, how modelers interact with them.

2.8 Conclusions and Future Work

This study investigates how novices and non-novices perform bug fixing tasks in Alloy models such that they match their natural language specifications. The results indicate that non-novices perform 54% better than novices on average and that participants perform better on syntactic tasks compared to semantic tasks. Non-novices spend less time working on the bug fixing tasks, and the participants in both groups use the action "Execute" most frequently while working on the Alloy models. The study results also show that small incremental changes are made before re-executing the model commands. The number of edits and actions performed is smaller with non-novices and predict accuracy in the novice group. This study has taken the critical first step towards digesting a practice that software designers have always engaged in, leading to an understanding that promises to enable researchers, practitioners, and educators to improve rigorous software modeling. As future work, we plan to qualitatively explore the participants' patterns of problem solving and study how Alloy novice and non-novices write specifications from scratch to describe a software system.

Chapter 3

An Empirical Assessment on Merging and Repositioning of Static Analysis Alarms

3.1 Introduction

Static analysis tools help to automatically detect common programming errors like *division by zero* and *array index out of bounds* [12, 78–82] and even certify absence of such errors in safety-critical systems [83–85]. However, these tools generate a large number of false positives [78, 86–89]. Previous studies report that there are 40 alarms for every thousand lines of code [90], and 35% to 91% of alarms are false positives [91]. Moreover, partitioning alarms into false positives and errors requires costly manual inspection [88, 89, 92]. Several studies [89, 90, 93, 94] 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 [92, 95–99]. 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 a technique, we consider *repositioning of alarms* [95, 96]. 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 3.2). The techniques implementing alarms repositioning are evaluated based on assumption that reporting a fewer alarms reduces the manual inspection effort. Hence, we investigate *to what extent repositioning of alarms improves user's performance during manual inspection of the alarms.* The reduction obtained is mainly due to two reasons: 1) merging multiple similar alarms, and 2) reporting alarms closer to their causes for their generation. Since the cognitive ability [44] 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 to each other - relevant in programming) correlated with their comprehension skills and the inspection accuracy.

This section of the dissertation 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.

3.2 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 [100].

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 this section, 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).

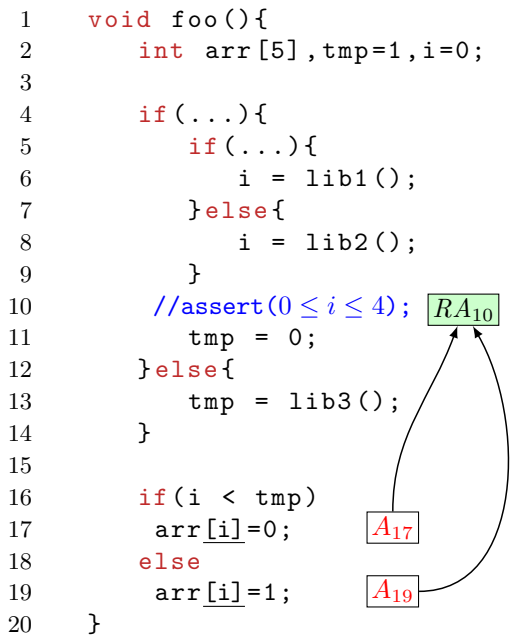


Figure 3.1: A code example showcasing repositioning of alarms.

We illustrate the two repositioning goals by borrowing and discussing the motivating example discussed in a prior study by Muske et al. [95] shown in Figure 3.1. 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

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

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}.

Postprocessing of alarms has mainly two goals: 1) to reduce the number of alarms, and 2) to simplify the manual inspection of alarms by simplifying the review process or providing review-assisting information. We observe that, among the 130 studies surveyed by Muske and Serebrenik [100] on techniques for postprocessing of alarms, only a few studies evaluate the techniques based on the reduction in inspection effort. Also, out of those 130 studies, 30 studies propose techniques that target simplification of manual inspection, and thereby reducing the manual inspection effort.

3.3 Research Questions and Hypotheses

3.3.1 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 3.2 the repositioning technique aims to achieve the effort reduction in two ways: (1) merging of alarms, and (2) reporting alarms closer to the cause point. Furthermore, it has been shown in prior literature that cognitive abilities such as amount of working memory and coding skills seem to be related [44]. In other work, it has been shown that mental rotation tasks [31] are also related to problem solving [45]. 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 two cognitive tasks we used are mental rotation [31] and operation span [32]. These are well defined questionnaires that we used directly from the psychology literature.

The following are the three research questions we are addressing in this study:

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?

RQ3. Is there a relationship between cognitive tasks and program comprehension tasks involving manual inspection of alarms?

3.3.2 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 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.

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.

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.

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.

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.

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.

3.4 Experiment Design

3.4.1 Study Overview

The goal of this controlled experiment [68,69] 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. We created an environment to study the behavior of participants while working on different types of tasks. Since we wanted to make adjustments on the presentation of alarms, either by repositioning or merging them, and we wanted to observe the effects of these adjustments in a setting that was controlled for different factors but was similar to a real-world setting, we decided that the controlled experiment was a suitable choice [69]. This type of experimental design is suggested when the behavior of users and participants is observed in a setting that is created specifically for the purpose of studying the effect of certain independent variables on dependent variables [69].

The study was hosted online on Qualtrics [101], a platform that allows building and distributing surveys. Qualtrics enables the 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) [102]. On the Mechanical Turk website, a requester can post a 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. We specified that the workers should be in the Software and IT Services, and stated in the study description that the participant should be familiar with the C language. 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.

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

Table 3.1: 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

3.4.2 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.

3.4.2.1 Cognitive Tasks

After 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 [32,71] and 2) 3D mental rotation task [31]. They measure spatial recognition ability and working memory respectively. Baum et al. [44] discovered that there is an effect from working memory capacity on bug localization accuracy in code reviews. Sharafi et al. [45] 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 [103] to build psychological experiments that run on the participants' browser. We used the open source Mental Rotation Task [104] and hosted it on a website which was linked in the survey. We also developed the operation span task using the magpie framework [105], 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.

3.4.2.2 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. [106] argue that it's important to include proficiency or screening questions in surveys, specifically in surveys done using online recruitment platforms. They mention their experience with running surveys and including questions for testing basic developer knowledge, and understanding that most of the participants did not understand a very simple program. Failure of adding proficiency questions to the experiment, will lead to inclusion of corrupt data from participants with no programming experience. We used the answers to these questions to assess the proficiency of the participants in C.

3.4.2.3 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 [107], 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 3.1 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 3.2. 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 3.2 for naming the tasks with their treatments.

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

Table 3.2: 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)

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.

3.4.3 Variables

The independent variables in this controlled experiment are repositioning and merging of alarms. Table 3.3 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

Table 3.3: 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

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.

3.4.4 Participants

We had two separate but identical Qualtrics surveys set up for participants recruited through Mechanical Turk and participants recruited through Link Share. We received 1254 submissions into our Link Share Qualtrics survey, and 141 entries into our Mechanical Turk Qualtrics survey. 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 3.5.1. After the pre-processing of the entries, we ended up with 151 entries from Link Share survey and 98 entries from Mechanical Turk survey.

3.4.5 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 of 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 the accepting and rejecting of submissions. After 20 workers completed the study, we examined their submissions and decided whether to accept or reject them. The survey was published to workers with the criteria of being employed in the Software and IT

Question: Does the assertion added in line 61 of "updating.h" hold?

(You can scroll through the code and the line with the assertion is highlighted. You can use Ctrl+F or Cmd+F to find the assertion by searching "assert". You can also move between the different files shown by clicking on the tab showing the file name.)

updating.h | archimedes.c | readinputfile.h

updating.h

```
47  // Monte Carlo Simulation
48  // ======================
49    if(model==MCE || model==MCEH){
50     EMC();
51     Charge();
52     if((int)(TF/DT)-(int)(TEMPO/DT)<MEDIA){
53       media();
54     }
55     if((TEMPO+DT)>=TF) DT=TF-TEMPO;
56     TEMPO+=DT;
57    }
58  // Electron MEP Simulation
59  // ======================
60    if(model==MEPE || model==MEPEH){
61     assert((ny+4)>=0 && (ny+4)<=308);
62     DT/=2.;
63     ParabMEP2D(nx,ny,dx,dy,0.475,1.0);
64     electron_relaxation_step();
65     DT*=2.;
66    }
67  // Hole MEP Simulation
68  // ===================
69    if(model==MEPH || model==MEPEH){
70     DT/=2.;
71     Hole_MEP2D(nx,ny,dx,dy,0.475,1.0);
72     Relaxation_Step_Hole();
73     DT*=2.;
74    }
```

Figure 3.2: An example question shown to the participants on Qualtrics

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 by 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 by 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 [108] 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. Figure 3.2 shows how a participant sees a task: the question asked and the C code files.

3.5 Experimental Results

3.5.1 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 3.4.4) 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 3.4: 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

Table 3.4 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 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 proficiency in the C programming language by setting criteria for 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 3.3 shows the differences between the level of experience of the participants of Link Share versus Mechanical Turk. Most of the participants from Link Share and Mechanical Turk think that they are 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 to our research questions. Table 3.5 shows how many entries were left after the cleanup of the data for each task in the LinkShare and Mechanical Turk surveys.

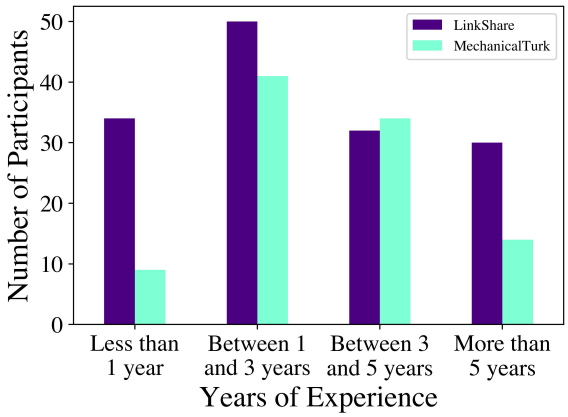


Figure 3.3: Years of experience in programming for Link Share and Mechanical Turk survey participants

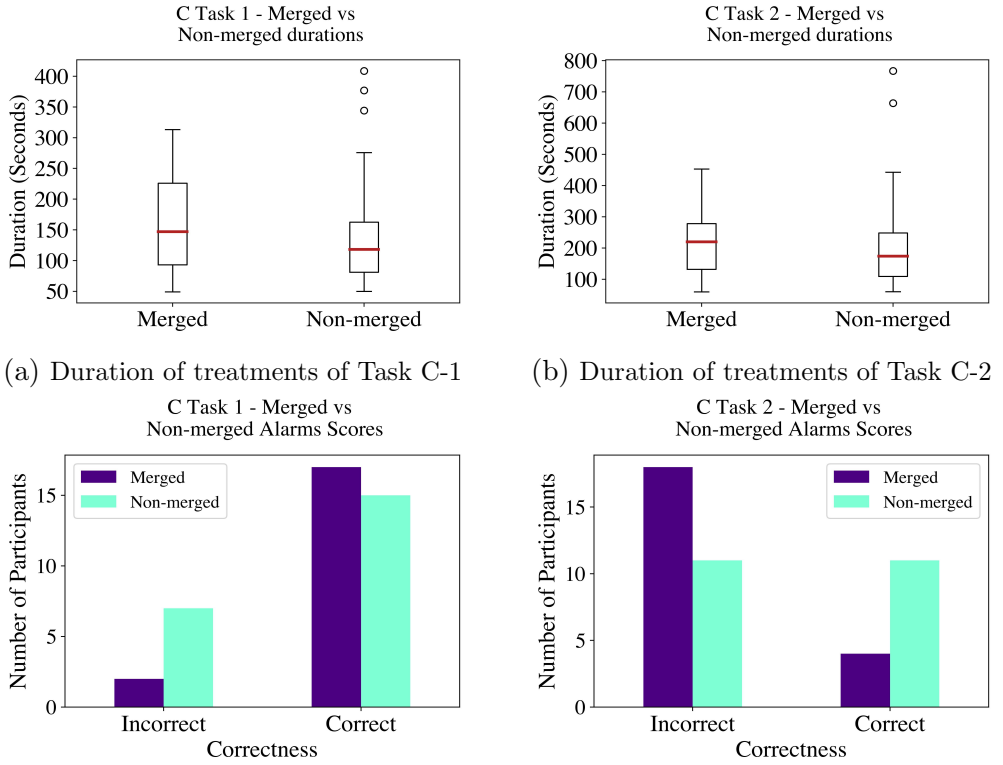
Table 3.5: Number of entries for each task

	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

3.5.2 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 3.4 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



(a) Duration of treatments of Task C-1

(b) Duration of treatments of Task C-2

(c) Accuracy for treatments of Task C-1

(d) Accuracy for treatments of Task C-2

Figure 3.4: Comparison of different treatments in the C task group

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-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 association between

the accuracy of answers and the 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 the 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.

3.5.3 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 3.5. 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

Table 3.6: Results of statistical tests for Task group B

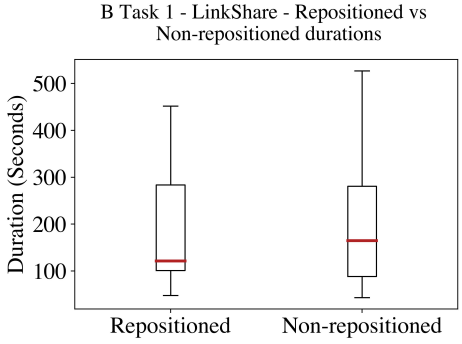
	Test	p	Effect Size	Repositioned Group N	Non-repositioned Group N
B-1 LinkShare	T-Test	0.889	-0.050	13	18
B-1 MTurk	Mann-Whitney	0.623	-0.104	8	12
B-2	Mann-Whitney	0.989	-0.168	26	16

T-Test or Mann-Whitney test based on the normality of data. Table 3.6 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: $p = 1$)

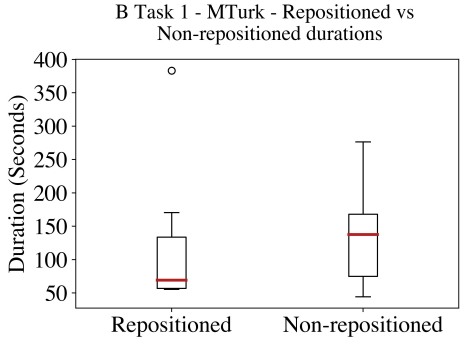
We cannot reject the null hypotheses CTH_0 and CAH_0, indicating lack of evidence that repositioning (closer-reporting) of alarms increases accuracy and speed of manual inspection.

3.5.4 RQ3 Results: Cognitive Tasks and Performance

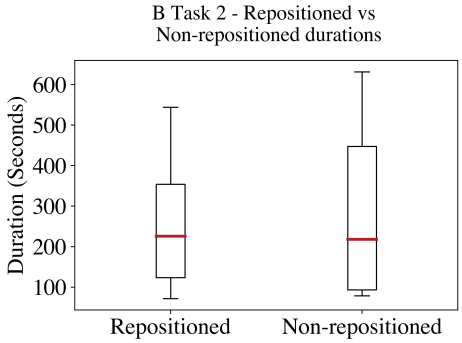
For RQ3, we looked at the relationship between the comprehension tasks accuracy score (C_{sco}) of the participants on the comprehension tasks to the scores they obtained on the two cognitive tasks. To find out which cognitive task entries were useful, we found the participants with proficiency score ≥ 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. ($N = 122$, $Mean_{C_{sco}} \pm SD = 1.93 \pm 1.22$)



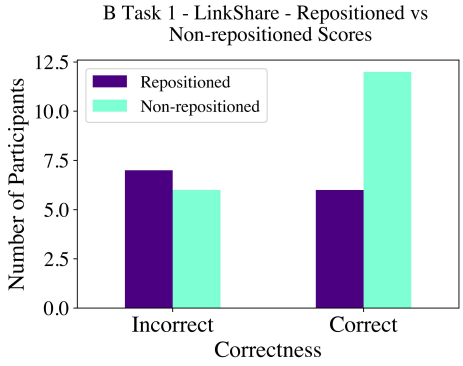
(a) Duration of treatments of Task B-1 - LinkShare



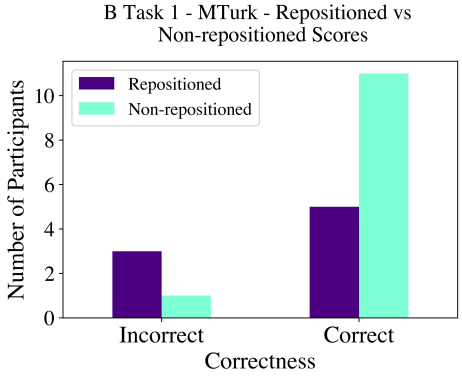
(b) Duration of treatments of Task B-1 - Mechanical Turk



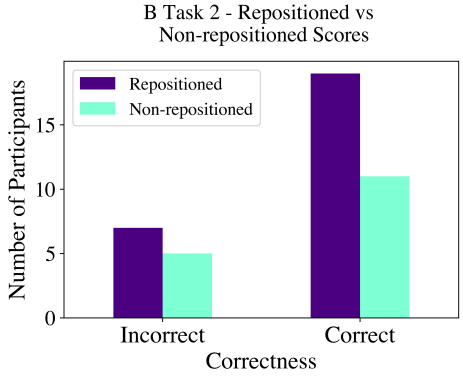
(c) Duration of treatments of Task B-2 (LinkShare and MTurk)



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



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



(f) Accuracy for treatments of Task B-2

Figure 3.5: Comparison of different treatments in the B task group

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 ($N = 111$, $Mean_{MRT_{sco}} \pm SD = 22.13 \pm 6.17$). In the Operation Span (OSPAN) Task, we calculated the score based on the operation span task MIS scoring method presented by Lammert et al. [109]. 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 ($N = 94$, $Mean_{OSPAN_{sco}} \pm SD = 6.077 \pm 1.66$).

To find the relationship between the cognitive scores and C_{sco}, we used Pearson's correlation. Pearson's correlation is used to examine the relationship between two quantitative variables and would tell us if one of the scores affects the other. We wanted to know how much the cognitive skills of a participant affect their performance on the comprehension tasks. 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 programmers' spatial abilities and their problem solving skills.

3.5.5 Post Questionnaire Results

Overall, most participants found the study somewhat difficult. Breaking this down further, out of the 122 participants with proficiency test scores ≥ 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 difficult", ten people believed it was "Somewhat easy", and four believed that it was "Extremely easy".

3.6 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 3.3). 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.

3.7 Discussion and Implications

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

3.7.1 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 a reduction, but it is negligible when compared with the effort required to inspect the repositioned alarms.

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

Next, we discuss these possibilities in detail.

3.7.1.1 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 [95], 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.

3.7.1.2 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 [90]. 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. Our study showed that it takes a considerable amount of time to work on the repositioning tasks, and the difference between time spent on non-treatment tasks versus treatment tasks was neither significant or consistent. This can indicate that the time required to inspect is considerable, and this effort dominates the effort that gets reduced.

3.7.1.3 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 the same task with and without 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.

In summary, the empirical evaluation results indicate that, in contrast to the expectations, the merging and repositioning of alarms do not reduce manual inspection effort and sometimes have a negative impact. A closer look at the results suggests that the study results are inconclusive and a more detailed study with selected experienced developers needs to be performed to evaluate the premise. In this proposed study, participants are monitored while working on the tasks and then interviewed. Eye

tracking is a more appropriate method to measure the time that participants spend on different code constructs and navigating the code. The next chapter of this dissertation presents the results of the eye tracking study.

3.7.2 Impact on Inspection Accuracy Improvement

As discussed earlier (Section 3.5), the merging and repositioning of alarms do not improve 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 about 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 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 the impact of merging and repositioning 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.

3.7.3 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. It is possible that working memory plays a role in solving the manual inspection tasks as well, as the developers might need to keep track of the value of the variables throughout the program. However, the operation span task scores and the survey scores were not correlated, and it might be due to the type of tasks we had for this specific study. 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 the accuracy of the comprehension tasks.

3.8 Conclusions and Future Work

The study investigates the effect of merging and repositioning of alarms on their inspection time and accuracy. We also explored the relationship between spatial cognitive skills and developers' comprehension skills. 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. Our analysis showed that the participants' spatial cognitive abilities correlated with their comprehension skills and accuracy of the tasks in this survey. This work provides an experience

and a methodology to conduct similar studies for evaluating alarm post-processing techniques. We have also discussed a few possibilities for the inconclusive results obtained for alarms merging and repositioning.

Chapter 4

Eye Tracking Study on Repositioning and Merging of Static Analysis Alarms

4.1 Introduction

In order to gather more insight into the results from the study conducted online presented in Chapter 3, we designed and conducted an eye tracking study that seeks to answer questions about code reading patterns and program comprehension while developers work on manually inspecting static analysis alarms. This eye tracking study might help us understand the effects of repositioning and merging static analysis alarms in tasks related to manual inspection of alarms in a more detailed manner by studying the developers' gaze and exploring their visual attention while they are solving such tasks. This eye tracking study serves as a complementary study to the online study discussed in Chapter 3, with the goal of understanding the effects of repositioning and merging by using eye tracking data that represents the developer's attention and focus while solving the tasks. This study explores effects we could not uncover only by measuring accuracy and speed in the online study.

In this chapter, we first explore some related work on eye tracking in program comprehension (Section 4.2). We then present the research questions and our hypotheses in Section 4.3 and the experimental design for the eye tracking study in

Section 4.4. We present the experiment results in Section 4.5, and we discuss the results in Section 4.7. Finally, Section 4.8 presents the conclusion to the study.

4.2 Related Work

Program comprehension is a cognitive process in which developers gain knowledge and understanding about a computer program [110]. Developers spend a lot of time reading code and understanding it, and this process has been studied throughout the last few decades, from papers studying the cognitive processes in program comprehension using verbal summaries of developers [111] to surveying different models of program comprehension [112] and reviewing theories and methods in program comprehension [113].

Program comprehension is difficult to study and measure as it involves complex cognitive processes. Recently, more studies have been done on program comprehension by using objective biometric methods such as eye tracking, EEG (electroencephalogram), and functional brain imaging techniques. For instance, Busjahn et al. [114] performed an eye tracking study and found that novices read Java code less linearly than natural language text, and compared to novices, experts read code in a less linear fashion. Fakhoury et al. [115] used eye tracking and brain imaging to study the effects of poor lexicon in source code on program comprehension and readability. Their results showed that poor naming and documentation practices increase the cognitive load in developers while solving tasks. For a more comprehensive review of the state of eye tracking studies done on program comprehension, we direct the reader to prior systematic literature reviews [116, 117].

Program comprehension studies have mostly been done with developers working on specific types of tasks such as bug fixing and summarization, and on more popular

programming languages such as Java, Python, and C++ (to mention a few: [22–28, 118–120]). Some studies have noted that the differences between experts and novices in program comprehension can be significant [121–123]. While many of these studies have been performed on debugging tasks, none of them have used helping methods such as static analysis to assist the developer while working on the tasks. The overall lack of literature on empirical studies of the usage of static analysis tools is mentioned in [15].

We also explore the relationship between program comprehension abilities and other cognitive processes, such as working memory and spatial cognition. Working memory refers to the temporary storage of information while performing other cognitive tasks such as reading, problem-solving, or learning [124]. It has direct effects on reading comprehension [125], and since programmers spend a lot of time reading code to understand and work with it, working memory capacity is an important metric for understanding how programmers complete difficult tasks. The working memory capacity was found to be associated with finding delocalized defects more effectively during code review [44]. They also found that the effectiveness of code reviews is significantly larger for small code changes. Another important cognitive ability is spatial cognition. Spatial intelligence is defined as the ability to form a mental model of the spatial world and being able to manipulate this model" [126], and there have been many studies revealing the relationship of spatial ability and success in mathematics and science [38]. One of the tests to measure spatial abilities is the Mental Rotation Test (a test requiring participants to mentally rotate three-dimensional solid objects), and Sharafi [45] used the test in their program comprehension study and found that spatial ability and data structure manipulation are correlated. We have asked the participants in this study to complete both working memory and mental rotation

tasks after the eye tracking tasks in the lab, with the goal of exploring the relationship between the developers' task accuracy score and their program comprehension skills.

4.3 Research Questions and Hypothesis

4.3.1 Research Questions

The goal of this eye tracking study is to analyze the developers' eye gaze data while they work on manual inspection of static analysis alarms and explore the effects of repositioning and merging the alarms on the developers' visual effort and attention. Visual effort is measured as the amount of visual attention allocated to parts of a visual stimulus, and we measure the visual attention by calculating different fixation metrics. These alarms are presented in the form of assertions on specific lines of code, representing the type of alarms a static analysis tool would generate for a developer. The developers work on multiple tasks in the C programming language that include one or more alarms, in which some of the alarms are in their original position, some are repositioned to be closer to the alarm source, and some are repositioned and merged. We capture the eye gazes on the code while the developers complete the comprehension tasks. We look at the *code traversals* that are performed by participants, which we define as how they look through the code to find relevant information to answer the tasks. Furthermore, we are interested in seeing the relationship between cognitive skills such as working memory and spatial recognition, and program comprehension skills. We use the information collected in our study to answer the following research questions:

RQ1. How are the code traversals performed by the developers during manual inspection of static analysis alarms?

RQ2. How does repositioning a single alarm affect accuracy and code traversals during manual inspection of static analysis alarms?

RQ3. How does repositioning and merging multiple alarms affect accuracy and code traversals during manual inspection of static analysis alarms?

RQ4. Is there a relationship between cognitive tasks and tasks involving manual inspection of alarms?

4.3.2 Hypotheses

Visual Effort And Accuracy Hypothesis (VEAH)

RQ1 explores how the developers perform code traversals when they are manually inspecting code to evaluate the static analysis alarms. For this research question, we explore the eye gaze patterns on the tasks that did not receive any repositioning or merging treatment, to understand how the developers read code and comprehend it in order to evaluate the assertions. In addition to exploring the reading and comprehension patterns, we have a hypothesis in regard to the accuracy and visual effort on relevant areas while solving the tasks. The null and alternate hypotheses are as follows:

$VEAH_0$ There is no significant difference in visual effort on relevant areas between participants who answered the questions accurately and inaccurately.

$VEAH_A$ There is increased visual effort on relevant areas from participants who answered the questions accurately compared to those who answered inaccurately.

Repositioning Effects Hypotheses (REH)

RQ2 explores the effect of repositioning on the code traversal and comprehension of the tasks. We can address questions about comprehension by measuring visual effort and the accuracy of answers.

The following hypothesis seeks to test whether repositioning a single alarm and moving it closer to the cause of alarm has an effect on the overall visual effort developers spend on relevant areas of the task. The null and alternate hypothesis are as follows:

$REHV_0$ Repositioning the alarms does not make a significant difference in visual effort on relevant areas.

$REHV_A$ Repositioning the alarms has an effect on visual effort on relevant areas.

The following hypothesis seeks to test if repositioning an alarm has an effect on the accuracy of the developers' answers. The null and alternate hypothesis are as follows:

$REHA_0$ Repositioning the alarms does not make a significant difference in accuracy.

$REHA_A$ Repositioning the alarms has an effect on accuracy.

Merging Effects Hypotheses (MEH)

RQ3 explores the effect of merging and repositioning multiple alarms on the code traversal and comprehension of the tasks. We can address questions about comprehension by measuring visual effort and accuracy. We present two related hypotheses that seek to test whether merging similar alarms has an effect on the overall visual effort developers spend on relevant areas of the task (MEHV), and if it has an effect on accuracy of the developers' answers (MEHA).

The null and alternate hypothesis about the visual effort effect are as follows:

$MEHV_0$ Merging multiple alarms does not make a significant difference in visual effort on relevant areas.

$MEHV_A$ Merging multiple alarms has an effect on visual effort on relevant areas.

The null and alternate hypothesis about the effect on accuracy are as follows:

$MEHA_0$ Merging multiple alarms does not make a significant difference in accuracy.

$MEHA_A$ Merging multiple alarms has an effect on accuracy.

Cognitive Tasks and Accuracy Hypothesis (CAH)

This hypothesis is related to RQ4, and seeks to test if there is a relationship between cognitive tasks score and the accuracy of manual inspection of alarms tasks.

CAH_0 There does not exist a relationship between accuracy on cognitive tasks and tasks involving manual inspection of alarms.

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

4.4 Experimental Design

4.4.1 Study Overview

The controlled experiment is similar to the online experiment discussed in Chapter 3, Section 3.4. The eye tracking study was also approved by the university's institutional review board. A quiet and interruption free room was available to conduct the study. During the eye tracking study, we used the Qualtrics platform again to display the pre-questionnaire, proficiency questions, manual inspection of alarm questions, links

to the cognitive tasks, and finally, the post-questionnaire. Similar to the online study, the randomization of the comprehension questions was done by Qualtrics. We used the Eclipse IDE for C/C++ Developers [127] to display the C files, including the comprehension tasks to the participants, and the iTrace Eclipse plugin [50, 128–130] to collect detailed line and token-level data from the participants' gazes on the source code. iTrace allows for seamless switching between different files and scrolling through the files, and it does not interfere with the participant's workflow while they are working on the tasks. A complete replication for the eye tracking study [1] is available for download.

4.4.2 Procedure

When the participants arrived at the location, they were asked to read a page containing an overview of the study and to sign a consent form. Next, the graduate student researcher explained the points discussed in the overview document in detail. The researcher explained the types of questions and gave an overview of how the eye tracking study will be conducted, and let the participants know about the constraints on their actions during the study (i.e., searching in the code was allowed, and editing the code was not.). The consent form and the study overview are available in Appendix B.

The researcher also explained the meaning of assertions and alarms to ensure that the participants were familiar with the concept. Participants were encouraged to ask questions to make sure that they understood what the study entails before beginning. They were then presented with the survey webpage and were asked to complete the pre-questionnaire and proficiency questions first. After completing these two categories of questions, a learning task was presented to the participant. At this step, the learning task question would appear on the screen, and the researcher

[1] `https://osf.io/92b7w/?view_only=5d1e162860b24d5badad350b2da2f598`

would familiarize the participant with the process of working on tasks in which their gazes are recorded. Eye tracker calibration is needed for each participant to record their gaze data properly, and the eye tracker gets calibrated several times during the study to ensure the preciseness of recorded data. The learning task was then shown to them on the Eclipse page. The participants were asked to let the researcher know once they were done with the task, and they were led back to the question page on Qualtrics and asked to choose "Yes" if the assertion in the question held, and "No" if the assertion did not hold. Subsequently, for each of the alarm inspection questions that were randomly chosen for the participant, the researcher would open the corresponding task file or folder on Eclipse, start the eye tracking session for that task, and ask the participant to answer the question on the Qualtrics survey once they were done with each task. After finishing the comprehension tasks, participants were presented with the links to the Mental Rotation and Working Memory cognitive tasks with a unique ID presented to them to enter into the cognitive tasks' webpage. The researcher explained the procedure of both tasks to the participants. After finishing the cognitive tasks, participants completed a post-questionnaire that asked them about their experience participating in the study.

4.4.3 Eye Tracking Apparatus

For the study, participants were seated in front of a 23-inch LCD monitor equipped with a Tobii TX300 eye tracker. The eye tracker was used with a sample rate of 60 Hz, and the screen resolution was set to 1920x1080. The code for the manual inspection tasks was shown in Eclipse For C, and the font size was set to 14 pt. The iTrace [50, 128–130] for Eclipse plugin was connected to the installed version of Eclipse to link the source code elements in IDE to the eye tracking data. A screenshot of the IDE connected to iTrace plugin is shown in Figure 4.1 Prior to collecting data

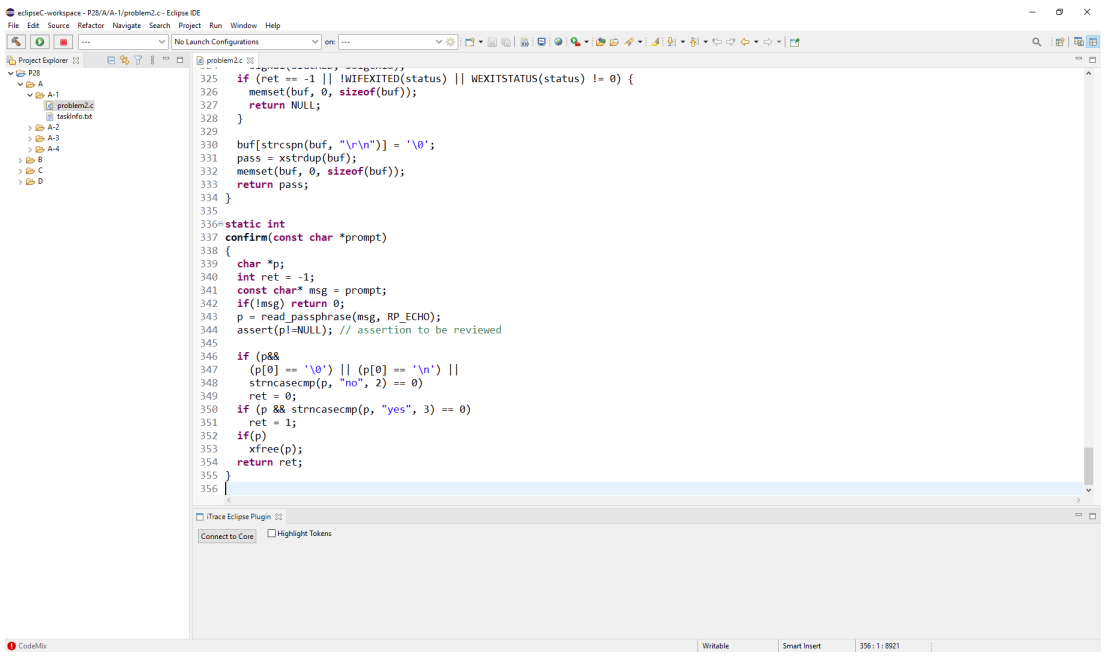


Figure 4.1: Screenshot of study environment showing the Eclipse IDE with C code and the i-Trace plugin running in the background.

for the study, we ran several tests to ensure the accuracy of the collected data. Even though iTrace supports tracking high-speed trackers [131], we only needed to track at 60Hz to answer our research questions. For more detailed saccade level analysis, higher frequency trackers can be used.

4.4.4 Participants

We advertised for the eye tracking study in our school's mailing list, and we recruited students and professionals through personal contacts. The advertisement on the mailing list [132] included a link to a form that asked the interested students about their age, experience with C, and their major and year in college. If they were under 19 years old or they did not have any experience with C programming, they were not qualified to do the study. The graduate student researcher followed up with the interested students who fit the criteria and scheduled a session with them to complete the study in the eye tracking lab. Through these methods, we were able to bring in

Table 4.1: Summary of demographic and experience details from the Eye Tracking study participants

	Choices	**Participants**
Age	19-23	11
	24-28	6
	29-33	8
	34-38	1
	39-43	0
	44-48	1
	49+	0
Gender	Woman	13
	Man	13
	Non-binary	1
	Undisclosed	0
Programming Experience Compared to Peers	Less experienced	10
	Equally experienced	15
	More experienced	2
Years of Programming Experience	Less than 1 year	0
	Between 1 and 3 years	6
	Between 3 and 5 years	8
	More than 5 years	13
Usage of Static Analysis Tools	Yes	9
	No	18
Frequency of Fixing Bugs	A couple times a year	5
	Every month	8
	Every week	7
	Everyday	7
Most Familiar IDE	Visual Studio	13
	Eclipse	7
	NetBeans	0
	Intellij	3
	Other	4
Years of Experience in Industry	0-1	18
	2-4	8
	5+	1

27 participants for the study. All participants were compensated with a gift card for their participation.

Table 4.1 presents the demographic data collected in the pre-questionnaire. We asked the participants about their age, gender, programming experience (asked in comparative terms as suggested by Siegmund et al. [70]), years of programming experience, usage of static analysis tools, frequency of bug fixing, the IDE they are most familiar with, and years of experience in industry.

Eleven participants were between the ages of 19-23, six participants were between the ages of 24-28, eight participants were between the ages of 29-33, one participant was between the ages of 34-38, and one participant was between 44-48 years old. Among the participants, thirteen identified as women, thirteen identified as men, and one participant identified as non-binary.

The participants answered some questions about their programming skills and abilities as well. Ten participants considered their programming experience to be less than their peers, twelve considered themselves equally experienced as their peers, and two considered themselves more experienced than their peers. Nine participants indicated that they have used static analysis tools before, while eighteen participants said that they did not use such tools before. When asked about the frequency of bug fixing, five participants indicated that they fix bugs a couple of times a year, while eight said that they fix bugs every month. Seven participants said that they fix bugs every week, and the remaining seven indicated that fix bugs everyday. We also asked the participants about the IDE they are most familiar with, and most of the participants were most familiar with Visual Studio. Finally, eighteen participants indicated that they have 0-1 years of experience working in industry, eight said that they have 2-4 years of experience in industry, and one said that they had more than five years of industry experience.

Table 4.2: Eye Tracking 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, Fixation Count, Fixation Duration, Average Fixation Duration, Revisits
Secondary Factors	Mental Rotation and Operation Span Scores

4.4.5 Study Variables

Our goal in the eye tracking study is to measure accuracy and visual effort to be able to answer the research questions and test our hypotheses. Accuracy refers to whether participants answered the task correctly. Since the answer to each task is either "Yes" or "No", we specify the correctness for each task with 1 as correct and 0 as incorrect.

We measure visual effort by different fixation metrics. Fixations are defined as "The stabilization of the eye on part of a stimulus for a period of time" [133]. There is a link between fixations and cognitive processes, as more focus on an area of interest indicates that the person is trying to understand and interpret what they are fixating on. Rayner [134] states that fixations can be as short as 50-75 milliseconds, and we specified fixations as gazes lasting more than 80 milliseconds for our study. We chose the following metrics to answer our questions about Visual Effort on specific areas of interest (AOIs):

- Fixation Count (FC) - The total number of fixations on an AOI
- Fixation Duration (FD) - The total time (ms) of all fixations on an AOI

- Average Fixation Duration (AFD) - The average fixation duration on an AOI
- Revisits - The number of times the participant visited a specific AOI

Higher fixation count, duration, and revisits on a specific AOI indicate that participants have put more effort into understanding the content of it. In our analysis, we focus on the eye movement metrics from gazes on specific "blocks" of the code for each task. These blocks are the relevant lines of code that the participant should focus on for answering the tasks.

For RQ1, we use the fixation metrics to determine how much participants focused on the relevant parts of the code to solve the tasks, and we use the fixation count and fixation duration on the AOIs of the code to determine the patterns of code traversals while solving the manual inspection tasks. We also compare the visual effort between participants who answered the tasks accurately and inaccurately. For RQ2 and RQ3, we compare the fixation metrics between the participants receiving different treatments to determine the effects of repositioning and merging on visual effort. Finally, for RQ4, we explore the relationship between task accuracy and cognitive tasks' scores.

4.4.6 Tasks and Areas of Interest

Table 4.3 shows the groups of tasks presented to each participant. Each participant worked on the proficiency tasks, the manual inspection of alarms tasks, and then finally the cognitive tasks. The order of tasks in the proficiency and manual inspection groups were randomized. To avoid repetition, we refer the reader to the sample randomization of tasks presented in Table 3.2 and an explanation of the order and randomization of tasks in Section 3.4.2.3. Section 3.4.2.1 presents more information about the cognitive tasks used in the eye tracking study.

Table 4.3: Summary of the task groups considered in the eye tracking 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**
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	RQ1, RQ4
	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	RQ2, RQ4
	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	RQ3, RQ4
Cognitive tasks	3D Mental Rotation Task	30	-	48	30	-	Random	RQ4
	Operation Span Task	10	-	-	10	-	Random	

In our study, the areas of interest (AOIs) over the source code of the eye tracking tasks are determined by creating "Blocks" for each task. A block is defined as a group of consecutive lines in a program that contain information that is relevant and needed for solving the task.

Each of the assertions in our tasks pose a question about the value of a variable. To determine the blocks for each task, we looked for lines of code that either directly assigned the value of the variable inside the assertion, or lines of code that had an indirect effect on the value. We also included the assertion lines as separate blocks in our analysis. As an example, if we saw an if-else statement that assigned a value to the variable in each of its branches, we specified the entire if-else statement as a block. Tables 4.4 and 4.5 present the categories of information contained in blocks and how they correspond to each block on the listed tasks. The *assertions* presented to the participants are always in one of the relevant blocks, as it is necessary for the participant to look at the assertion to answer the question. If the variable that is asked about in the assertion or another related variable to the task is assigned somewhere in the code, it's relevant to the task. Thus, we have the *Variable Assignment* category in some of the blocks. Sometimes a variable's definition is important to answer a task, and in those cases we have *Variable Definition* in some of our blocks. Some of the blocks contain *Method Signatures*, *Method Body*, or *Method Calls* and these methods are the ones necessary to look through to understand some information about the value of the variables important to the task. And finally, some of the blocks contain *Error Handling and Value Checks*, which are blocks of code that generate errors based on some specific values or check values to stay in specific ranges. We have also specified which blocks are relevant for solving different treatments of the tasks in the table.

Table 4.4: Block descriptions for group A task files

Task	File	Block	Type	Relevant in Task
A-1	problem2.c	1	Assertion, Variable Assignment, Method Call	A-1
		2	Method Signature	
		3	Method Body	
		4	Method Signature	
		5	Method Body	
		6	Method Body	
A-2	problem4.c	1	Assertion, Variable Assignment	A-2
		2	Method Body	
		3	Method Body	
		4	Variable Definition	
A-3	problem5.c	1	Assertion	A-3
		2	Variable Definition	
		3	Variable Assignment	
		4	Method Call	
		5	Method Signature	
A-4	problem6.c	1	Assertion	A-4
		2	Error Handling and Value Checks	
		3	Error Handling and Value Checks	
		4	Variable Assignment	
		5	Method Call	
		6	Error Handling and Value Checks	
		7	Method Signature, Method Body	

In the following section, we discuss one of the tasks in detail and we present our rationale for specifying the relevant blocks for that task. The reader can refer to Appendix B for the rest of the tasks and their specified blocks.

An Example Task and AOIs

Listing 4.1 shows the relevant blocks of code for task B-1. Task B-1 was presented to participants either without treatment (with the single assertion on line 122), or with the repositioning treatment (single assertion on line 40). The assertion asks the

Table 4.5: Block descriptions for group B and C task files

Task	File	Block	Type	Relevant in Task
B-1	file_orig.c/file_repos.c	1	Assertion (Original), Method Signature, Method Body	B-1-A
		2	Assertion (Treatment), Variable Assignment, Method Call	B-1-A, B-1-B
		3	Method Body	
		4	Method Signature, Method Body	B-1-A
		5	Method Signature, Method Body	
		6	Method Signature, Method Body	
		7	Method Signature, Method Body	
B-2	file_orig.c/file_repos.c	1	Assertion (Original), Method Signature, Variable Assignment	B-2-A
		2	Method Signature, Variable Assignment, Method Call	
		3	Method Signature, Variable Assignment, Method Call	
		4	Method Signature, Method Call	
		5	Assertion (Treatment), Variable Assigmment, Method Signature	B-2-A, B-2-B
C-1	file_orig.c/file_repos.c	1	Assertion (Original), Method Signature	C-1-A
		2	Assertion (Original), Method Signature	
		3	Assertion (Original), Method Signature	
		4	Assertion (Original), Method Signature	
		5	Assertion (Original), Method Signature	
		6	Assertion (Treatment), Variable Assignment	C-1-A, C-1-B
		7	Method Body, Variable Assignment	
C-2	Hole_bcs.h	1	Assertion (Original)	C-2-A
		2	Assertion (Original)	
		3	Method Signature	
	Holemep2d.h	1	Method Call	
		2	Method Signature	
	HMEPbcs.h	1	Assertion (Original)	
		2	Assertion (Original)	
		3	Method Signature	
	ParabMEP2D.h	1	Method Call	
		2	Method Signature	
	updating.h	1	Assertion (Treatment), Method Call	C-2-A, C-3-B
		2	Method Call	
		3	Method Signature	
	archimedes.c	1	Method Call	
		2	Method Call	
		3	Variable Definition	
	readinputfile.h	1	Variable Definition	
		2	Error Handling and Value Checks	

participant about the value of `index`, and whether `index` $\geq$ `0` or `index` $\leq$ `63`. The participant should answer "yes" if they believe that `index` is within that range, and "no" if they think the value is outside of the specified range in the assertion.

In the non-treatment version of the task, the assertion is placed inside a method called `unlock_rules`. We specify Block 1 as the method signature, the lines surrounding the assertion, and the assertion line itself. We expect the developer to trace the global variable `index` throughout the code to determine its value at line 122.

Block 2 starts from the initial value of `index`, and it includes the next number of lines that contain information about the value of `index`. As seen in the code, the variable `index` is incremented inside a while loop, and it gets incremented until the loop exits. The exit condition of the while loop depends on the values of `buflen` and `MAX_BUFLEN`. `MAX_BUFLEN` is defined as 1024 in the code. To find the value of `buflen`, the participant has to look at method `handle_cmdline(int do_detach)`. Which is why we have specified that method as Block 3. We observe that this method prevents its return value to be greater than `MAX_BUFLEN`, and the largest value it returns is `MAX_BUFLEN - 1`. After determining the value of `buflen`, the participant might look at the method `acpid_add_client(int clifd, const char *origin)` to trace its effect on index and the flow of the program. This method is specified as Block 4. Block 4 contains a line that calls the method `parse_client(int client)`. We specify this method as Block 5. Inside Block 5, a method named `enlist_rule(struct rule_list *list, struct tule *r)` is called, which we name as Block 6. Next, we see that `do_client_rule(struct rule *rule, const char *event)` is called inside of Block 6. We specify this method as Block 7 in this task. And finally, the method that contains the assertion (`unlock_rules(void)`) is called from Block 7.

All of these methods specified as different blocks are called after one another to eventually execute `unlock_rules`, which contains the assertion. Any of the blocks

could have had an effect on the value of `index` while it was flowing through the program to get to line 122. But after looking through these methods we realize that none of these methods have changed the value of `index`. The participant can then go back to Block 2 with the understanding that `index` is only incremented on line 49 of the code. The participant proceeds to observe that `buflen` is bound by the value of `MAX_BUFLEN` and is multiplied by 2 in each iteration of the loop. Which means that if we consider the lowest (1) and highest (1023) values that can be assigned to `buflen`, we can see that the index will at most be incremented to 10. With this information, the participant can determine the answer to the assertion, which is "Yes".

This is just one possible example traversal to find the value of `index` at line 122. Different programmers might approach the problem differently and look at the different sections in different order. But the importance of these blocks is clear in finding the answer to the question.

In the repositioned treatment, by moving the assertion to line 40 and before calling the `acpid_add_client(int clifd, const char *origin)` method, we draw the focus to Block 2 and Block 3 which include the information needed to solve the task. With the repositioned alarm, visiting the functions is not a priority for the participants and they can determine the value of index in a more efficient way.

Listing 4.1: The relevant blocks for task B-1.
Block numbers for each block are specified at the beginning of each block.

```
      1  /*
      2  Does the assertion included on line 122 (line 40) hold?
      3  */
      4
      5  #define NULL 0
      6  #define MAX_BUFLEN 1024
      7  int buffer[64],index,acpid_debug;
Block 2  ...
     35          index = 0;
     36          buflen = handle_cmdline(do_detach);
     37          while (1){
     38                  if (do_detach) {
     39                          /* tell our clients to buzz off */
     40                          assert(index >= 0 && index <= 63); (Repositioned Alarm)
     41                          acpid_add_client(do_detach,p->origin);
     42                          p = client_list.head;
     43                          while (p) {
     44                                  next = p->next;
     45                                  close(p->action.fd);
     46                                  p = next;
     47                          }
     48                  }
     49                  index ++;
     50                  if (buflen >= MAX_BUFLEN) {
     51                          break;
     52                  }
     53                  buflen *= 2;
     54          }
Block 4  ...
     64  int
     65  acpid_add_client(int clifd, const char *origin)
     66  {
     67          struct rule *r;
     68          int nrules = 0;
     69
     70          r = parse_client(clifd);
     71          if (r) {
     72                  r->origin = strdup(origin);
     73                  nrules++;
     74          }
     75          return 0;
     76  }
Block 3  ...
     78  int handle_cmdline (int do_detach){
```

```
      79          if (!do_detach) {
      80                  printf("ERR: malloc(%d): %s\n");
      81          }
      82
      83          if(do_detach >= MAX_BUFLEN || do_detach < 0)
      84                  return MAX_BUFLEN - 1;
      85          else
      86                  return do_detach;
      87  }
Block 6   ...
      89  static void
      90  enlist_rule(struct rule_list *list, struct rule *r)
      91  {
      92          if (!list->head) {
      93                  list->head = r;
      94                  list->tail = r;
      95          } else {
      96                  list->tail->next = r;
      97                  list->tail = r;
      98          }
      99          do_client_rule(r,r->origin);
     100  }
Block 5   ...
     102  static struct rule *
     103  parse_client(int client)
     104  {
     105          struct rule *r;
     106          int rv;
     107
     108          if (rv) {
     109                  char buf[128];
     110                  regerror(rv, r->event, buf, sizeof(buf));
     111          }
     112          enlist_rule(&client_list, r);
     113          return r;
     114  }
Block 1   ...
     116  static void
     117  unlock_rules(void)
     118  {
     119          if (acpid_debug >= 4) {
     120                  acpid_log("unblocking signals for rule lock\n");
     121          }
     122          assert(index >= 0 && index <= 63); (Original Alarm)
     123          acpid_log(buffer[index]);
     124  }
Block 7   ...
     126  static int
     127  do_client_rule(struct rule *rule, const char *event)
     128  {
     129          int r;
```

```
130         int client = rule->action.fd;
131
132         if (r < 0 ) {
133                 /* closed */
134                 close(rule->action.fd);
135                 free_rule(rule);
136         }
137         unlock_rules();
138         return 0;
139 }
```

4.5 Experimental Results

4.5.1 Data Pre-processing

We used iTrace [50] for the real-time mapping of source code elements to participants' eye gazes. iTrace processes the gazes captured by the eye tracker, and checks whether the gazes fall on specific UI widgets in Eclipse. The i-Trace Eclipse Plugin maps the coordinates to specific files, lines, and columns of the source code. The software then exports each gaze event and the attached information to XML files. The XML file for each task includes the eye gaze events on the source code along with the information about the X and Y coordinates of the gaze, left and right pupil diameters, the time of the gaze reported by both the eye tracker and the system, the viewed files, and the viewed lines for each gaze. We wanted to generate token level fixations for our study, to get the most accurate fixations on different source code elements. To do so, we ran SrcML [135] on the source code of the tasks to have a representation of the source code elements with tags that specify the various elements of the abstract syntax in XML format.

iTrace generates a pair of XML files for each task completed, and each pair was processed using the iTrace Toolkit to export the session and gaze information into databases. We created one database for each task that included all the sessions from

all of our participants. We then used the toolkit's ability to identify tokens under gazes using the SrcML files for the programs and the file, column, and line information from the iTrace files. After identifying the tokens, we ran the IVT fixation filter with the velocity threshold of 50 and duration of 80 milliseconds. The generated fixations for the tasks provided us with the necessary data for running our scripts. We developed various scripts to 1) identify the correspondent block for each fixation, 2) calculate the fixation count, fixation duration, average fixation duration, and revisits metrics, 3) identify the number of files visited and the metrics on each file for the tasks that presented multiple files to the participants. We developed scripts to calculate the cognitive task scores based on the formulas mentioned in Chapter 3, section 3.5.4.

4.5.2 Statistical Tests

We used the JASP statistical software [76] to perform our statistical tests. To answer RQ1, RQ2, and RQ3 we needed to compare two independent groups. Whether the two groups are created by dividing the participants with different accuracy scores for a task, or if they are divided by receiving a treatment or non-treatment task, our groups are always independent of one another. To choose the suitable statistical test for comparing the two independent groups, we first run the normality (Shapiro-Wilk test) and then homogeneity of variance (Levene test) checks. If the groups pass the normality and the homogeneity of variance tests, we perform independent samples t-test to compare the means of the groups, and calculate Cohen's d to report effect size. If the groups pass the normality test but not the homogeneity of variance test, we use Welch's t-test to compare the means of the groups. And finally, if the groups do not pass the normality check, we perform Mann-Whitney U test, which is a non-parametric alternative test to independent samples t-test. Mann-Whitney U test's effect size is calculated by rank-biserial correlation. The significance level is a

threshold determining if the results are considered statistically significant. It is set as a p-value that is less than 0.05.

4.5.3 RQ1: Patterns of Solving Manual Inspection Tasks

To answer RQ1 and test hypothesis VEAH, we investigated tasks from Task Group A - Manual Inspection Of Alarms. The 27 participants each randomly received two of the four available tasks from group A. Fourteen participants answered task A-1, nine participants answered task A-2, fourteen participants answered task A-3, and seventeen participants answered task A-4. We generated the following metrics for tasks A-1, A-2, A-3, and A-4: Fixation Count (FC), Fixation Duration (FD), Average Fixation Duration (AFD), and Revisits over the specified blocks on the code of each task. Each of the tasks in the A group asked the participant to evaluate one assertion on only one file. None of the tasks required the participant to look at multiple files.

To test hypothesis VEAH, we compared the fixation metrics between the participants who answered each task correctly and the ones who answered incorrectly. We wanted to know if there were any specific parts of the code that were more important for accurately answering the tasks. Furthermore, in order to explore the problem solving patterns, we look at the participants' fixations on each block and how they read the program to solve each tasks. We visualize the fixation patterns over time by generating a scarf plot for each task using Alpscarf [136]. Scarf plots are useful to display patterns of AOI visits, and we use them to understand how the participants have looked at the different blocks on each task.

Table 4.6: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration) for relevant blocks of task A-1
The number of lines for each block are specified.

	Metric	Incorrect													Correct				
Participant / Block		**P01**	**P04**	**P07**	**P09**	**P11**	**P12**	**P13**	**P15**	**P17**	**P19**	**P25**	**Mean**	**Std. Dev.**	**P02**	**P23**	**P24**	**Mean**	**Std. Dev.**
Block 1	FC	48	67	294	118	383	80	112	104	112	30	420	160.73	137.58	30	103	157	63.74	96.67
(7 Lines)	FD	22775	45889	195153	49083	175616	37107	46774	55979	46783	20376	171652	78835.18	66604.13	7782	54851	124678	58816.06	62437
	AFD	474.47	684.91	663.78	415.95	458.52	463.83	417.62	538.26	417.71	679.2	408.7	511.18	112.19	259.4	532.53	794.13	267.39	528.69
	Revisits	9	17	53	19	49	11	18	14	24	5	46	24.09	17.07	7	12	37	16.07	18.67
Block 2	FC	20	24	8	20	27	19	27	33	53	6	24	23.73	12.56	6	56	5	29.16	22.33
(2 Lines)	FD	10587	13229	1416	7486	7767	5519	6396	14798	16974	1614	8138	8538.55	5007.85	2952	20046	1162	10424.45	8053.33
	AFD	529.35	551.2	177	374.3	287.66	290.47	236.88	448.42	320.26	269	339.08	347.6	118.57	492	357.96	232.4	129.82	360.79
	Revisits	5	15	4	10	12	11	14	19	25	4	20	12.64	6.87	6	14	5	4.93	8.33
Block 3	FC	114	168	3	154	212	219	88	247	400	23	233	169.18	112.51	79	267	347	137.58	231
(23 Lines)	FD	54246	77542	4599	61944	88050	128039	36035	113436	146717	12029	75751	72580.73	45373.83	25678	117253	242642	108920.28	128524.33
	AFD	475.84	461.55	1533	402.23	415.33	584.65	409.48	459.26	366.79	523	325.11	541.48	336.58	325.03	439.15	699.26	191.8	487.81
	Revisits	6	17	2	14	11	11	8	30	41	5	26	15.55	12.06	18	21	32	7.37	23.67
Block 4	FC	0	4	3	3	3	5	2	10	7	1	19	5.18	5.36	0	13	12	7.23	8.33
(2 Lines)	FD	0	1965	599	1550	965	1700	785	3632	4711	1083	6123	2101.18	1911.66	0	5668	4252	2949.88	3306.67
	AFD	0	491.25	199.66	516.66	321.66	340	392.5	363.2	673	1083	322.26	427.56	277.84	0	436	354.33	231.77	263.44
	Revisits	0	4	1	3	3	4	2	9	7	1	11	4.09	3.51	0	8	6	4.16	4.67
Block 5	FC	2	4	1	2	0	35	3	6	4	0	23	7.27	11.22	3	12	8	4.51	7.67
(14 Lines)	FD	617	3816	217	850	0	24795	1932	1531	2020	0	9389	4106.09	7371.26	634	6185	5862	3115.82	4227
	AFD	308.5	954	217	425	0	708.42	644	255.17	505	0	408.22	402.3	292.59	211.33	515.42	732.75	261.91	486.5
	Revisits	2	4	1	2	0	3	3	5	3	0	17	3.64	4.7	3	7	6	2.08	5.33
Block 6	FC	0	17	27	6	9	5	8	29	13	1	42	14.27	13.26	15	3	6	6.25	8
(9 Lines)	FD	0	6663	10431	1365	4715	1665	1534	13480	3914	750	14421	5358	5217.17	4867	1267	3226	1802.34	3120
	AFD	0	391.94	386.33	227.5	523.88	333	191.75	464.83	301.08	750	343.36	355.79	192.72	324.46	422.33	537.67	106.72	428.15
	Revisits	0	10	16	5	3	3	5	11	10	1	32	8.73	9.12	10	2	3	4.36	5

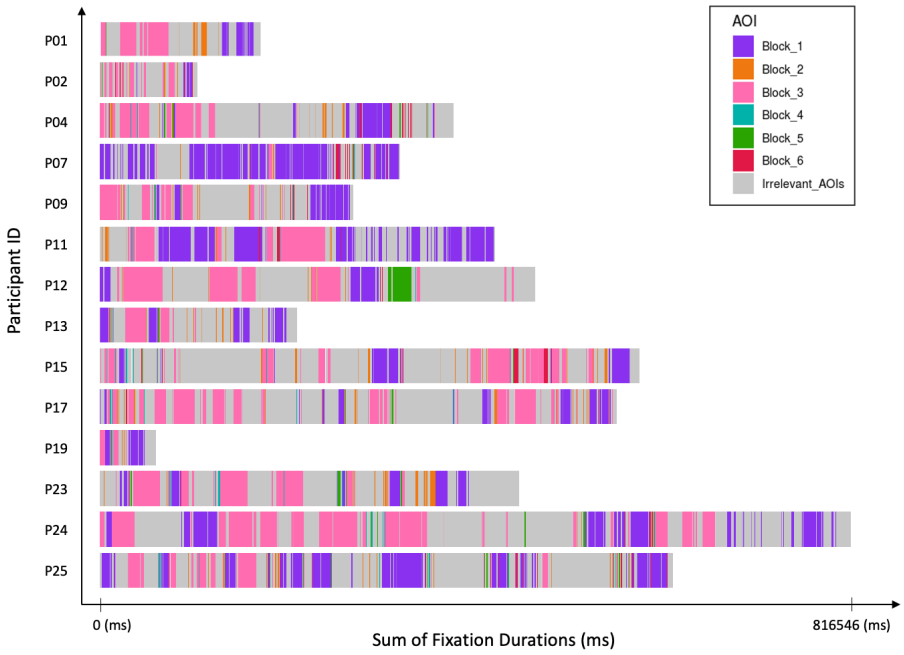


Figure 4.2: Scarf plot of A-1

Results for A-1: Manual Inspection of Alarms.

Table 4.6 shows the metrics related to fixations on the six different blocks for participants, categorized by the correctness of the task. The mean and standard deviation of the metrics in each group is reported in the table. We compared the fixation metrics for each block between the two groups. Table 4.7 shows the statistical tests and the results of comparing the fixation metrics from each block. None of the tests show any significant differences between the accurate/inaccurate groups in task A-1.

Figure 4.2 shows the scarf plot of task A-1. The scarf plot shows that the most visited blocks for task A-1 were Block 3 and Block 1, with participants going back and forth between these two blocks after looking at irrelevant code to the task (specified by Irrelevant AOIs in the plot). Block 3 contains lines that return a value from a method, which is directly called from Block 1 to assign a value to the variable inside

Table 4.7: Statistical tests to compare metrics from A-1 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Mann-Whitney	20.5		0.585	0.242
(7 Lines)	FD	Mann-Whitney	18		0.885	0.091
	AFD	Mann-Whitney	15		0.885	-0.091
	Revisits	Mann-Whitney	21		0.555	0.273
Block 2	FC	Mann-Whitney	21.5		0.482	0.303
(2 Lines)	FD	Student's t-test	0.119	12	0.907	0.078
	AFD	Student's t-test	-0.168	12	0.869	-0.109
	Revisits	Student's t-test	1.002	12	0.336	0.653
Block 3	FC	Student's t-test	-0.811	12	0.433	-0.528
(23 Lines)	FD	Student's t-test	-1.413	12	0.183	-0.921
	AFD	Mann-Whitney	18		0.885	0.091
	Revisits	Student's t-test	-1.092	12	0.296	-0.712
Block 4	FC	Mann-Whitney	12.5		0.584	-0.242
(2 Lines)	FD	Student's t-test	-0.873	12	0.4	-0.569
	AFD	Student's t-test	0.931	12	0.37	0.606
	Revisits	Student's t-test	-0.244	12	0.811	-0.159
Block 5	FC	Mann-Whitney	9.5		0.309	-0.424
(14 Lines)	FD	Mann-Whitney	11		0.436	-0.333
	AFD	Student's t-test	-0.449	12	0.661	-0.293
	Revisits	Mann-Whitney	6.5		0.134	-0.606
Block 6	FC	Student's t-test	0.779	12	0.451	0.507
(9 Lines)	FD	Student's t-test	0.713	12	0.489	0.464
	AFD	Student's t-test	-0.613	12	0.551	-0.399
	Revisits	Mann-Whitney	21		0.529	0.273

the assertion. Five participants first went to the assertion and fixated on the block that includes the assertion, and the rest either started with Block 3 or Block 2 (which is the method signature of the method contained in Block 3).

Table 4.8: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration) for relevant blocks of task A-2

	Metric	**Incorrect**						**Correct**						
Participant / **Block**		**P06**	**P18**	**P20**	**P26**	**Mean**	**Std. Dev.**	**P01**	**P05**	**P10**	**P12**	**P21**	**Mean**	**Std. Dev.**
Block 1	FC	50	63	150	112	93.75	46.03	9	24	25	53	56	33.4	20.31
(6 Lines)	FD	34891	30765	85768	85462	59221.5	30523.36	4767	11048	15500	29900	32378	18718.6	11994.24
	AFD	697.82	488.33	571.79	763.05	630.25	123.51	529.67	460.33	620	564.15	578.18	550.47	59.89
	Revisits	14	27	17	13	17.75	6.4	6	6	11	11	17	10.2	4.55
Block 2	FC	130	176	31	18	88.75	76.71	26	18	50	23	93	42	31.06
(9 Lines)	FD	53191	67688	9948	7223	34512.5	30537.58	8914	7947	22557	6764	29369	15110.2	10224.13
	AFD	409.16	384.59	320.9	401.28	378.98	40.05	342.85	441.5	451.14	294.09	315.8	369.08	72.68
	Revisits	25	35	13	10	20.75	11.5	6	8	10	14	18	11.2	4.82
Block 3	FC	113	40	31	16	50	43.15	18	3	20	77	33	30.2	28.24
(6 Lines)	FD	54105	13995	9816	6447	21090.75	22224.98	10450	1098	7719	54290	11850	17081.4	21207
	AFD	478.81	349.88	316.65	402.94	387.07	70.74	580.56	366	385.95	705.06	359.09	479.33	155.9
	Revisits	33	24	12	6	18.75	12.09	8	3	6	20	8	9	6.48
Block 4	FC	34	70	77	38	54.75	21.9	12	0	10	41	37	20	17.99
(4 Lines)	FD	11484	27235	38378	12811	22477	12777.04	3415	0	4800	14893	10016	6624.8	5860.55
	AFD	337.76	389.07	498.42	337.13	390.6	75.89	284.58	0	480	363.24	270.7	279.7	177.09
	Revisits	10	26	25	5	16.5	10.6	6	0	8	12	7	6.6	4.34

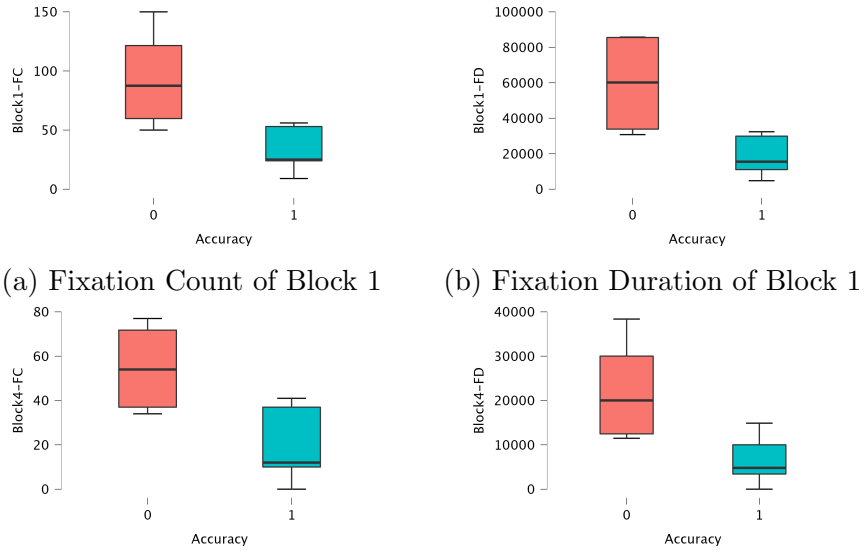
Table 4.9: Statistical tests to compare metrics from A-2 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Student's t-test	2.66	7	0.032*	1.785
(6 Lines)	FD	Student's t-test	2.752	7	0.028*	1.846
	AFD	Student's t-test	1.283	7	0.24	0.861
	Revisits	Student's t-test	2.077	7	0.076	1.393
Block 2	FC	Student's t-test	1.257	7	0.249	0.843
(9 Lines)	FD	Student's t-test	1.349	7	0.219	0.905
	AFD	Student's t-test	0.243	7	0.815	0.163
	Revisits	Student's t-test	1.702	7	0.132	1.142
Block 3	FC	Student's t-test	0.834	7	0.432	0.559
(6 Lines)	FD	Mann-Whitney	11		0.905	0.1
	AFD	Student's t-test	-1.086	7	0.313	-0.729
	Revisits	Student's t-test	1.561	7	0.162	1.047
Block 4	FC	Student's t-test	2.622	7	0.034*	1.759
(4 Lines)	FD	Student's t-test	2.497	7	0.041*	1.675
	AFD	Student's t-test	1.158	7	0.285	0.777
	Revisits	Student's t-test	1.923	7	0.096	1.29

Results for A-2: Manual Inspection of Alarms

Table 4.8 shows the fixation metrics for the relevant blocks for task A-2, divided into two groups by accuracy, as well as the mean and standard deviation of metrics for each group. Table 4.9 shows the statistical tests between the two groups. The t-test shows a significant difference between the fixation count ($p = 0.032$, ES = 1.785) and fixation duration ($p = 0.028$, ES = 1.846) on Block 1, as well as a significant different between the fixation count ($p = 0.034$, ES = 1.759) and fixation duration ($p = 0.041$, 1.675) on Block 4. Figure 4.3 shows the box plots representing the significantly different metrics in two groups of accuracy (0: inaccurate, 1: accurate).

Figure 4.4 shows the scarf plot of task A-2. The scarf plot suggests the five out of nine participants focused on Block 1 at first, which contained the assertion. The plot also shows that most of the blocks fixated on by the participants were relevant blocks to the task. Block 2 and Block 3, which contained statements that determined the



(a) Fixation Count of Block 1 (b) Fixation Duration of Block 1

(c) Fixation Count of Block 4 (d) Fixation Duration of Block 4

Figure 4.3: Box plots of A-2 metrics with significant differences in two groups

value of the variable inside the assertion were visited frequently by the participants. Both the statistical tests and the scarf plot show clear differences between fixation count and duration over Block 1 and Block 4 between accurate (Participants P1, P5, P10, P12, and P21) and inaccurate groups (P6, P18, P20, P26), with participants in the inaccurate group fixating more and longer on these two blocks.

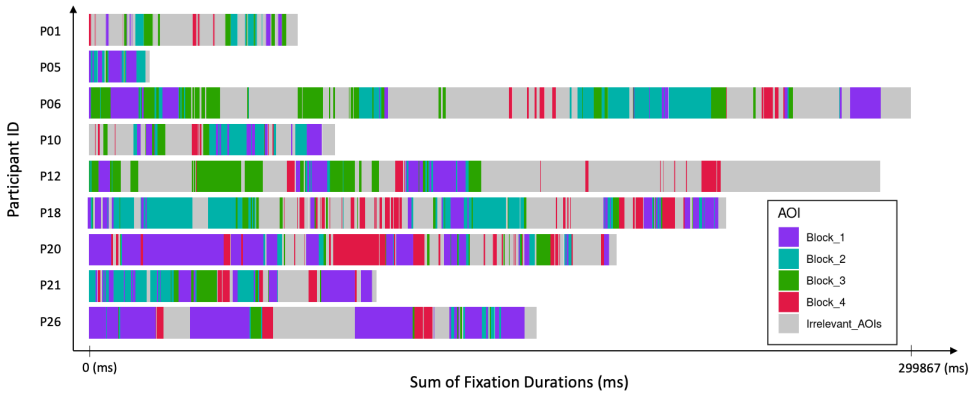


Figure 4.4: Scarf plot of A-2

Table 4.10: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration) for relevant blocks of task A-3

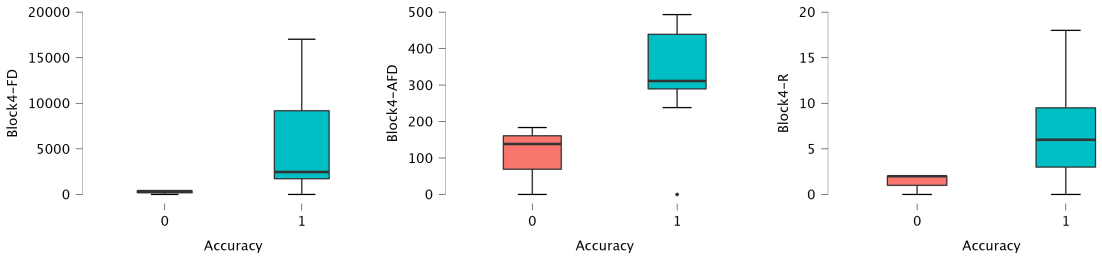
	Metric	Incorrect					Correct												
		P16	**P17**	**P21**	**Mean**	**Std. Dev.**	**P03**	**P04**	**P05**	**P07**	**P08**	**P11**	**P13**	**P22**	**P26**	**P27**	**P28**	**Mean**	**Std. Dev.**
Block 1	FC	48	228	53	109.67	102.51	103	48	122	51	45	83	159	106	238	43	214	110.18	68.25
(3 Lines)	FD	25063	93299	20460	46274	40789.83	35364	26923	44609	35355	19199	31475	46164	41239	118746	35213	66926	45564.82	27181.66
	AFD	522.15	409.21	386.04	439.13	72.82	343.34	560.9	365.65	693.24	426.64	379.22	290.34	389.05	498.93	818.91	312.74	461.72	167.49
	Revisits	11	33	5	16.33	14.74	24	13	12	11	14	20	26	17	13	3	25	16.18	7.03
Block 2	FC	57	74	31	54	21.66	63	19	0	0	11	5	34	22	21	8	119	27.45	35.34
(9 Lines)	FD	34164	31943	11320	25809	12596.89	27427	12582	0	0	3799	2633	14268	7768	10378	4048	44874	11616.09	13598.99
	AFD	599.37	431.66	365.16	465.4	120.69	435.35	662.21	0	0	345.36	526.6	419.65	353.09	494.19	506	377.09	374.5	206.18
	Revisits	11	13	4	9.33	4.73	10	8	0	0	5	1	8	1	6	1	18	5.27	5.57
Block 3	FC	10	96	84	63.33	46.58	82	22	0	150	2	15	175	48	456	42	266	114.36	141.23
(5 Lines)	FD	2466	33128	28048	21214	16433.72	34694	8366	0	74205	650	5279	59896	13071	177070	20027	77251	42773.55	53069.73
	AFD	246.6	345.08	333.9	308.53	53.92	423.1	380.27	0	494.7	325	351.93	342.26	272.31	388.31	476.83	290.42	340.47	132.86
	Revisits	7	15	6	9.33	4.93	11	9	0	27	2	7	27	8	44	7	37	16.27	14.89
Block 4	FC	0	2	3	1.67	1.53	3	37	7	30	5	6	42	15	7	0	21	15.73	14.65
(2 Lines)	FD	0	367	415	260.67	227.02	1267	17027	2082	13679	2466	1783	11830	5114	1667	0	6534	5768.09	5812.3
	AFD	0	183.5	138.33	107.28	95.61	422.33	460.19	297.43	455.97	493.2	297.17	281.67	340.93	238.14	0	311.14	327.11	137.9
	Revisits	0	2	2	1.33	1.15	3	10	6	9	3	3	18	10	6	0	9	7	4.96
Block 5	FC	3	7	1	3.67	3.06	17	13	0	20	2	3	49	6	16	4	16	13.27	13.78
(4 Lines)	FD	1201	1950	150	1100.33	904.21	3919	7080	0	10151	500	499	14377	1884	4742	2434	5179	4615	4474.67
	AFD	400.33	278.57	150	276.3	125.18	230.53	544.62	0	507.55	250	166.33	293.41	314	296.38	608.5	323.69	321.36	175.8
	Revisits	3	2	1	2	1	6	7	0	12	2	2	21	3	6	4	10	6.64	5.95

Table 4.11: Statistical tests to compare metrics from A-3 blocks ((FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Mann-Whitney	16.5		1	0
(3 Lines)	FD	Mann-Whitney	12		0.555	-0.273
	AFD	Student's t-test	-0.223	12	0.828	-0.145
	Revisits	Student's t-test	0.026	12	0.979	0.017
Block 2	FC	Mann-Whitney	27		0.119	0.636
(9 Lines)	FD	Mann-Whitney	27		0.119	0.636
	AFD	Student's t-test	0.717	12	0.487	0.467
	Revisits	Student's t-test	1.146	12	0.274	0.747
Block 3	FC	Mann-Whitney	16		1	-0.03
(5 Lines)	FD	Mann-Whitney	14		0.769	-0.152
	AFD	Mann-Whitney	10		0.368	-0.394
	Revisits	Student's t-test	-0.775	12	0.453	-0.505
Block 4	FC	Student's t-test	-1.612	12	0.133	-1.05
(2 Lines)	FD	Mann-Whitney	2.5		0.035*	-0.848
	AFD	Student's t-test	-2.561	12	0.025*	-1.668
	Revisits	Mann-Whitney	2.5		0.034*	-0.848
Block 5	FC	Mann-Whitney	8.5		0.242	-0.485
(4 Lines)	FD	Student's t-test	-1.316	12	0.213	-0.857
	AFD	Student's t-test	-0.411	12	0.688	-0.268
	Revisits	Student's t-test	-1.306	12	0.216	-0.851

Results for A-3: Manual Inspection of Alarms

Table 4.10 shows the fixation metrics for the relevant blocks for task A-3. The table is divided into two groups by accuracy, and the mean and standard deviation of metrics for each group is shown as well. Table 4.11 presents the statistical tests perform to explore the differences between the two groups. The tests show significant differences between fixation duration (p = 0.035, ES = -0.848), Average Fixation Duration (p = 0.025, ES = -1.668), and revisits (p = 0.034, ES = -0.848) of Block 4. Figure 4.5 shows the box plots representing the significant differences on Block 4 metrics between the two groups.



(a) Fixation Duration in Block 4

(b) Average Fixation Duration in Block 4

(c) Revisits of Block 4

Figure 4.5: Box plots of A-3 metrics with significant differences in two groups

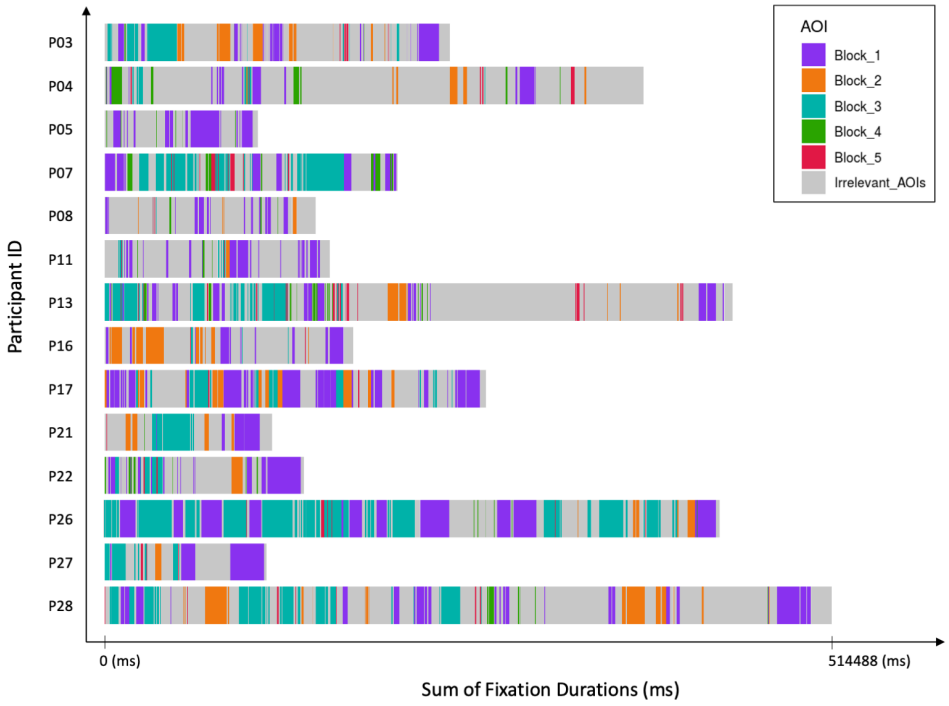
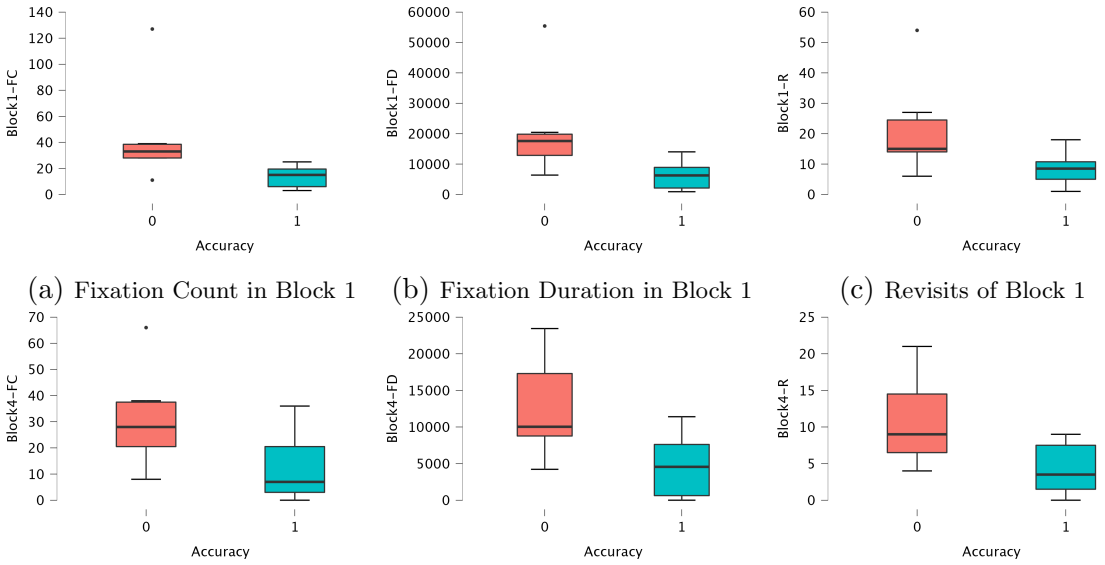


Figure 4.6: Scarfplot of A-3

Figure 4.6 shows the scarf plot of task A-3. The scarf plot shows that all participants have revisited Block 1 (the block containing the assertion) many times. The scarf plot shows that the participants in the inaccurate group (Participants P16, P17, P21) have not fixated on Block 4 for a significant amount of time compared to the accurate group. We can also observe that the participants have spent a lot of time fixating on irrelevant AOIs for this task. Another observation is that participants moved their attention from Block 1 to Block 3 and vice versa a number of times. Block 3 contains the important number of lines in this task that assign a value to the variable under assertion.

Table 4.12: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration) for relevant blocks of task A-4

	Metric	Incorrect									Correct											
Participant / Block		**P06**	**P15**	**P19**	**P22**	**P23**	**P24**	**P25**	**Mean**	**Std. Dev.**	**P02**	**P03**	**P08**	**P09**	**P10**	**P16**	**P18**	**P20**	**P27**	**P28**	**Mean**	**Std. Dev.**
Block 1	FC	28	39	11	28	38	33	127	43.43	38.01	18	15	22	5	3	15	9	25	5	20	13.7	7.79
(3 Lines)	FD	17583	19214	6369	10618	15095	20434	55433	20678	16108.04	9282	4790	14016	900	1218	9099	2952	8330	1834	7728	6014.9	4346.54
	AFD	627.96	492.67	579	379.21	397.24	619.21	436.48	504.54	104.82	515.67	319.33	637.09	180	406	606.6	328	333.2	366.8	386.4	407.91	140.45
	Revisits	13	27	6	15	15	22	54	21.71	15.72	1	10	14	5	3	9	8	18	5	11	8.4	5.17
Block 2	FC	29	55	8	18	74	43	88	45	29.29	10	42	18	35	7	58	17	102	11	28	32.8	29.18
(5 Lines)	FD	12417	23002	4834	5530	30880	31570	39533	21109.43	13729.92	2700	23580	8250	15911	2997	27604	5102	43515	5935	9869	14546.3	13298.92
	AFD	428.17	418.22	604.25	307.22	417.3	734.19	449.24	479.8	142.17	270	561.43	458.33	454.6	428.14	475.93	300.12	426.62	539.55	352.46	426.72	94.97
	Revisits	10	18	2	12	19	25	27	16.14	8.78	7	16	6	14	2	31	6	46	4	14	14.6	13.88
Block 3	FC	74	177	18	70	118	115	352	132	108.87	36	80	66	44	27	86	40	204	67	118	76.8	52.45
(12 Lines)	FD	24568	66632	8380	23859	53206	70725	139294	55237.71	43894.14	8907	32865	31684	18295	12235	36423	12984	75121	36078	48135	31272.7	20069.01
	AFD	332	376.45	465.56	340.84	450.9	615	395.72	425.21	97.78	247.42	410.81	480.06	415.8	453.15	423.52	324.6	368.24	538.48	407.92	407	80.88
	Revisits	18	43	5	18	31	38	71	32	21.6	15	22	19	12	5	32	14	41	12	24	19.6	10.64
Block 4	FC	66	8	28	19	22	38	37	31.14	18.59	3	24	0	3	9	5	36	0	13	23	11.6	12.24
(4 Lines)	FD	23443	4218	13602	8517	10029	20994	9024	12832.43	7011.69	532	9382	0	916	5480	3631	11401	0	7952	6600	4589.4	4196.69
	AFD	355.2	527.25	485.79	448.26	455.86	552.47	243.89	438.39	106.7	177.33	390.92	0	305.33	608.89	726.2	316.69	0	611.69	286.96	342.4	249.45
	Revisits	21	4	9	10	6	19	7	10.86	6.57	3	6	0	1	4	3	8	0	9	8	4.2	3.39
Block 5	FC	22	22	0	8	8	57	15	18.86	18.62	12	24	18	14	7	15	0	35	0	23	14.8	10.94
(1 Line)	FD	7496	7968	0	1400	3484	27079	3767	7313.43	9190.9	2951	8502	4968	3916	1851	6597	0	9833	0	8831	4744.9	3613.08
	AFD	340.73	362.18	0	175	435.5	475.07	251.13	291.37	164.33	245.92	354.25	276	279.71	264.43	439.8	0	280.94	0	383.96	252.5	146.26
	Revisits	13	13	0	4	3	23	8	9.14	7.86	9	13	8	9	4	11	0	32	0	8	9.4	9.05
Block 6	FC	17	13	3	9	13	90	42	26.71	30.51	2	17	16	36	9	47	0	61	1	56	24.5	23.55
(3 Lines)	FD	6065	3205	966	3783	3730	55925	15471	12735	19612.54	251	4632	5767	14832	3102	17550	0	18480	400	23018	8803.2	8749.13
	AFD	356.76	246.54	322	420.33	286.92	621.39	368.36	374.62	122.62	125.5	272.47	360.44	412	344.67	373.4	0	302.95	400	411.04	300.25	136.27
	Revisits	6	9	3	3	10	23	18	10.29	7.61	2	12	7	16	2	25	0	37	1	16	11.8	12.09
Block 7	FC	170	39	32	153	201	295	172	151.71	91.96	116	281	6	278	20	303	86	229	48	378	174.5	134.36
(21 Lines)	FD	72946	15963	13488	64394	90155	199974	59334	73750.57	62541.28	34071	128351	1749	123872	8685	125589	36800	97513	30126	140259	72701.5	55194.88
	AFD	429.09	409.31	421.5	420.88	448.53	677.88	344.97	450.31	105.45	293.72	456.77	291.5	445.58	434.25	414.49	427.91	425.82	627.63	371.06	418.87	94.45
	Revisits	23	8	6	11	14	19	14	13.57	5.97	12	13	3	10	5	17	9	10	4	15	9.8	4.69



(a) Fixation Count in Block 1 (b) Fixation Duration in Block 1 (c) Revisits of Block 1

(d) Fixation Count in Block 4 (e) Revisits of Block 4 (f) Fixation Duration of Block 4

Figure 4.7: Box plots of A-4 metrics with significant differences in two groups

Results for A-4: Manual Inspection of Alarms

Table 4.12 presents the fixation metrics on the relevant blocks for task A-4, along with the mean and standard deviation of each accuracy group. Table 4.13 reports on the statistical tests used to compare the fixation metrics from each block. The tests show significant differences between Block 1 Fixation Count ($p = 0.005$, ES $= 0.829$), Fixation Duration ($p = 0.003$, ES $= 0.829$), and Revisits ($p = 0.024$, ES $= 1.242$). They also show significant differences between Block 4 Fixation Count ($p = 0.019$, ES $= 1.294$), Fixation Duration ($p = 0.008$, ES $= 1.499$), and Revisits ($p = 0.015$, ES $= 0.1.354$). Figure 4.7 shows the box plots visualizing the significant differences between these metrics.

Figure 4.8 presents the scarf plot of task A-4. The scarf plot shows that Block 7, which contains lines that assign a new value to the variable under assertion in Block 1, is visited and revisited by all participants, and has been fixated on longer compared to all the other blocks. Block 3, which contains a number of lines that use the value

Table 4.13: Statistical tests to compare metrics from A-4 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Mann-Whitney	64		0.005*	0.829
(3 Lines)	FD	Mann-Whitney	64		0.003*	0.829
	AFD	Student's t-test	1.539	15	0.145	0.759
	Revisits	Student's t-test	2.52	15	0.024*	1.242
Block 2	FC	Mann-Whitney	45.5		0.329	0.3
(5 Lines)	FD	Student's t-test	0.988	15	0.339	0.487
	AFD	Student's t-test	0.927	15	0.369	0.457
	Revisits	Mann-Whitney	42.5		0.494	0.214
Block 3	FC	Mann-Whitney	47.5		0.241	0.357
(12 Lines)	FD	Student's t-test	1.528	15	0.147	0.753
	AFD	Student's t-test	0.42	15	0.681	0.207
	Revisits	Student's t-test	1.577	15	0.136	0.777
Block 4	FC	Student's t-test	2.626	15	0.019*	1.294
(4 Lines)	FD	Student's t-test	3.042	15	0.008*	1.499
	AFD	Student's t-test	0.952	15	0.356	0.469
	Revisits	Student's t-test	2.748	15	0.015*	1.354
Block 5	FC	Student's t-test	0.567	15	0.579	0.28
(1 Line)	FD	Mann-Whitney	35		1	0
	AFD	Student's t-test	0.513	15	0.615	0.253
	Revisits	Mann-Whitney	35.5		1	0.014
Block 6	FC	Mann-Whitney	37		0.883	0.057
(3 Lines)	FD	Mann-Whitney	38		0.813	0.086
	AFD	Mann-Whitney	40		0.669	0.143
	Revisits	Student's t-test	-0.292	15	0.774	-0.144
Block 7	FC	Student's t-test	-0.388	15	0.704	-0.191
(21 Lines)	FD	Student's t-test	0.037	15	0.971	0.018
	AFD	Mann-Whitney	37		0.887	0.057
	Revisits	Student's t-test	1.461	15	0.165	0.72

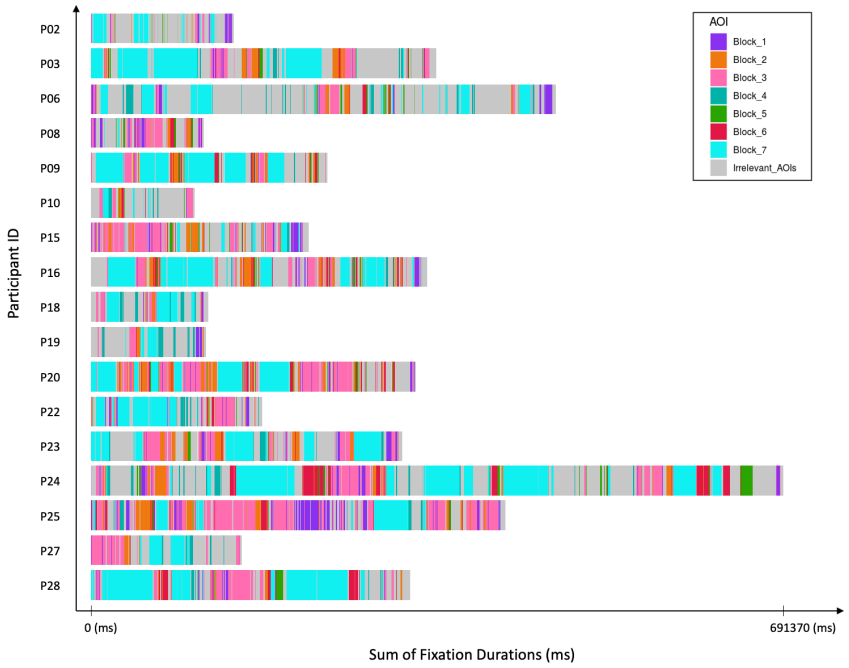


Figure 4.8: Scarf plot of A-4

of the variable under assertion, is also visited by all the participants. The plot also shows that many participants kept going back and forth between Block 7 and the other blocks.

Our findings from the tasks indicate that we do not have conclusive evidence to reject the null hypothesis $VEAH_0$. Despite the fact that we found significant differences between the metrics of some blocks on tasks A-2, A-3, and A-4, we did not find any significant differences between the metrics of the task A-1 blocks. Our results regarding the comparison of visual effort in accurate and inaccurate groups are inconclusive.

4.5.4 RQ2: Effects of Repositioning A Single Alarm

To answer RQ2 and test hypotheses $REHV$ and $REHA$, we explored the tasks from Task Group B - Repositioning of an Alarm. Each of the 27 participants received two

tasks from task group B, task B-1, and B-2. They randomly received one of the tasks without any treatment (i.e. the alarm in its original location) and the other task with the repositioning treatment (i.e. the repositioned alarm). The tasks without treatment are specified with A at the end of their name (e.g. B-1-A) whereas the tasks with repositioning treatment are specified with B (e.g. B-2-B). For Task Group B tasks, the following fixation metrics from the participants are generated: Fixation Count (FC), Fixation Duration (FD), Average Fixation Duration (AFD), and Revisits. Each task in the B group asked the participants to inspect and evaluate one assertion over one file.

To test hypothesis $REHV$ and to understand the effect of repositioning on visual effort, we looked at the fixation metrics for the entire file and then each block of the programs and compared them across the treatment and non-treatment groups. We also generated scarf plots showing the fixation patterns over the code for the participants in both of the groups for each task. To test hypothesis $REHA$ and explore whether repositioning of alarms has an effect on accuracy, we compared the accuracy of the participants' answers between the treatment and non-treatment groups. The answer to each question about an assertion is either "Yes" or "No" and we denote the correctness of the answers as 1 or 0 in our analysis.

Results for B-1: Repositioning of a Single Alarm

Tables 4.15 and 4.16 show the fixation metrics for the seven relevant blocks of task B-1. Out of the 27 participants, 14 participants received the task without repositioning (B-1-A), and 13 received the repositioned alarm task (B-1-B). For the analysis for B-1, we looked at the differences between the sum of fixation count and fixation duration over the entire file for the task, to see if the repositioning treatment has affected the metrics. The Welch's t-test shows that even though both the sum of fixation count

Table 4.14: Statistical tests to compare metrics from B-1 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Mann-Whitney	182		<.001*	1
(9 Lines)	FD	Mann-Whitney	182		<.001*	1
	AFD	Mann-Whitney	117		0.215	0.286
	Revisits	Welch's t-test	6.812	15.716	<.001*	2.583
Block 2	FC	Student's t-test	-1.227	25	0.231	-0.473
(20 Lines)	FD	Student's t-test	-1.514	25	0.142	-0.583
	AFD	Mann-Whitney	78		0.55	-0.143
	Revisits	Student's t-test	-0.661	25	0.515	-0.254
Block 3	FC	Mann-Whitney	85		0.789	-0.066
(10 Lines)	FD	Mann-Whitney	83.5		0.734	-0.082
	AFD	Mann-Whitney	71.5		0.356	-0.214
	Revisits	Student's t-test	-0.563	25	0.579	-0.217
Block 4	FC	Mann-Whitney	72		0.369	-0.209
(13 Lines)	FD	Mann-Whitney	75		0.452	-0.176
	AFD	Mann-Whitney	86		0.827	-0.055
	Revisits	Mann-Whitney	83.5		0.733	-0.082
Block 5	FC	Mann-Whitney	130.5		0.058	0.434
(13 Lines)	FD	Mann-Whitney	124		0.115	0.363
	AFD	Mann-Whitney	93		0.942	0.022
	Revisits	Welch's t-test	3	15.295	0.009*	1.137
Block 6	FC	Mann-Whitney	120.5		0.157	0.324
(12 Lines)	FD	Mann-Whitney	122		0.138	0.341
	AFD	Student's t-test	1.039	25	0.309	0.4
	Revisits	Mann-Whitney	105		0.508	0.154
Block 7	FC	Mann-Whitney	145		0.009*	0.593
(14 Lines)	FD	Mann-Whitney	150		0.004*	0.648
	AFD	Mann-Whitney	119		0.181	0.308
	Revisits	Mann-Whitney	157		0.001*	0.725

and sum of fixation duration over the entire file for the non-treatment task are higher in average compared to the repositioned treatment task, but there is no significant difference between the sum of fixation count ($p = 0.061$, ES = 0.429) and sum of fixation duration ($p = 0.061$, ES = 0.429) over the entire file.

The assertion shown to the participants is in Block 1 in task B-1-A, and the repositioning of the alarm in task B-1-B moves the alarm from Block 1 to Block 2. This repositioning is done with the goal of drawing the attention of the developer to Block 2 and Block 3, in which they can find the answer to the task. The tables show

the mean and standard deviation of the metrics in each block as well. We compared the fixation metrics for each block between the two groups. Table 4.14 shows the results of statistical tests performed on the metrics from each block. The results show a significant difference between Block 1 Fixation Count ($p = < 0.001$, ES = 1), Fixation Duration ($p = < 0.001$, ES = 1), Revisits ($p = < 0.001$, ES = 2.583), Block 5 Revisits ($p = 0.009$, ES = 1.137), and Block 7 Fixation Count ($p = 0.009$, ES = 0.593), Fixation Duration ($p = 0.004$, ES = 0.648), and Revisits ($p = 0.001$, ES = 0.725). Figure 4.9 shows the box plots of the significantly different metrics, which indicate that visual effort on Block 1 and Block 7 was higher and Block 5 had more revisits in the non-treatment task (B-1-A).

Table 4.15: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task B-1-A (non-treatment)

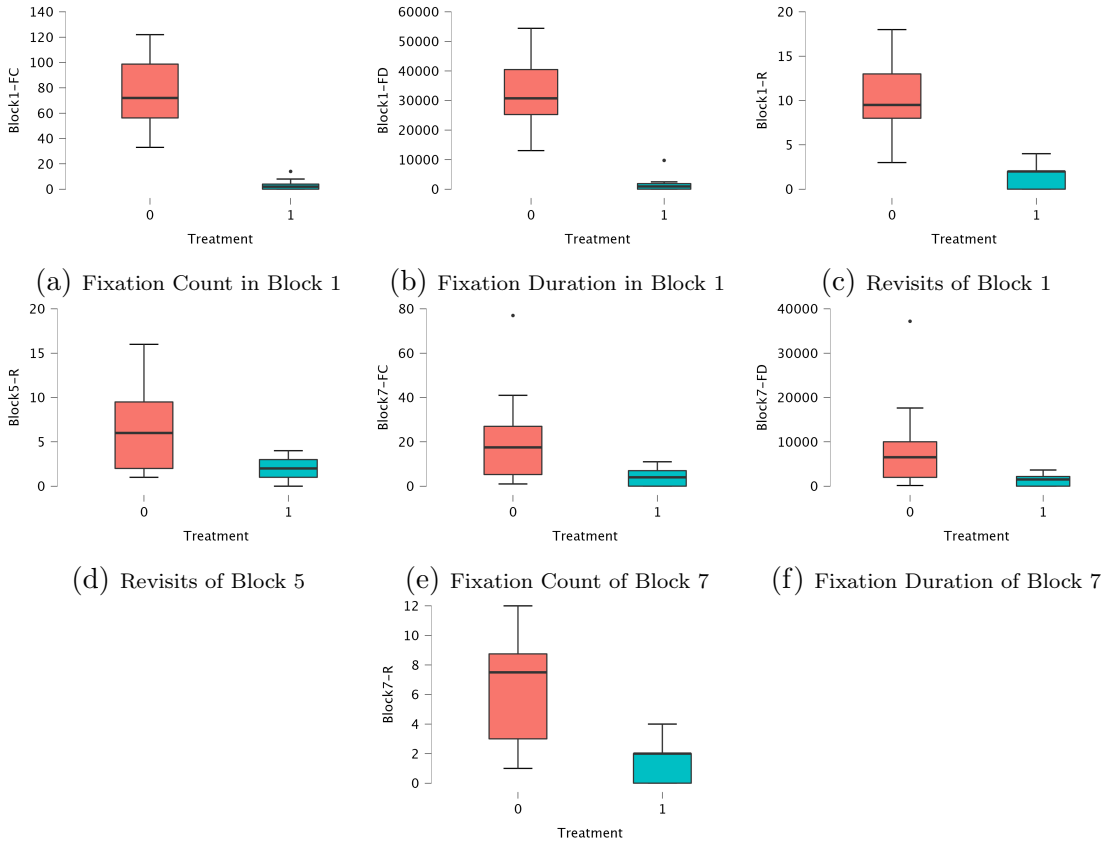
	Metric	Non-treatment															
PID / **Block**		**P01**	**P05**	**P06**	**P07**	**P10**	**P12**	**P13**	**P15**	**P17**	**P21**	**P22**	**P25**	**P26**	**P27**	**Mean**	**Std. Dev.**
Block 1	FC	33	122	100	69	68	54	116	77	95	49	75	103	60	55	76.86	26.67
(9 Lines)	FD	17782	54434	40261	31336	27496	21282	46823	31445	40549	13048	30148	42488	27928	24508	32109.14	11594.26
	AFD	538.85	446.18	402.61	454.14	404.35	394.11	403.65	408.38	426.83	266.29	401.97	412.5	465.47	445.6	419.35	58.42
	R	8	18	13	3	13	12	18	9	10	3	14	9	8	8	10.43	4.6
Block 2	FC	165	7	202	44	64	272	554	16	243	11	427	413	427	38	205.93	187.58
(20 Lines)	FD	84330	2298	99280	24346	23278	141968	212449	5029	104422	5547	178422	179302	194454	26245	91526.43	78231.91
	AFD	511.09	328.29	491.49	553.32	363.72	521.94	383.48	314.31	429.72	504.27	417.85	434.15	455.4	690.66	457.12	99.33
	R	25	4	19	11	15	32	85	7	30	3	44	59	57	4	28.21	24.95
Block 3	FC	83	7	110	0	18	69	186	0	142	0	155	6	303	23	78.71	91.37
(10 Lines)	FD	35433	2368	53617	0	5916	30075	70730	0	49383	0	61096	2370	142223	13297	33322	40301.13
	AFD	426.9	338.29	487.43	0	328.67	435.87	380.27	0	347.77	0	394.17	395	469.38	578.13	327.28	188.97
	R	17	5	6	0	2	10	13	0	5	0	9	3	19	4	6.64	6.18
Block 4	FC	17	9	4	1	4	67	65	0	6	0	11	26	39	7	18.29	22.94
(13 Lines)	FD	7465	2732	1983	267	1967	45514	20513	0	1013	0	3167	17855	13806	2382	8476	12660.39
	AFD	439.12	303.56	495.75	267	491.75	679.31	315.58	0	168.83	0	287.91	686.73	354	340.29	344.99	208.06
	R	10	5	3	1	3	14	17	0	6	0	7	11	19	2	7	6.29
Block 5	FC	7	26	18	1	21	34	112	4	4	2	16	22	11	3	20.07	28.36
(13 Lines)	FD	2950	8298	6083	899	7193	14464	42404	898	1599	333	6110	9528	2840	882	7462.93	10866.73
	AFD	421.43	319.15	337.94	899	342.52	425.41	378.61	224.5	399.75	166.5	381.88	433.09	258.18	294	377.28	169.87
	R	3	15	6	1	7	12	16	2	2	1	8	10	6	1	6.43	5.2
Block 6	FC	6	10	9	3	9	31	61	0	4	0	13	22	13	5	13.29	16.13
(12 Lines)	FD	2568	3348	2768	1318	4182	20864	28117	0	1083	0	6452	5645	6838	1097	6020	8254.73
	AFD	428	334.8	307.56	439.33	464.67	673.03	460.93	0	270.75	0	496.31	256.59	526	219.4	348.38	191.21
	R	3	4	3	2	5	10	16	0	2	0	6	7	9	2	4.93	4.41
Block 7	FC	5	23	27	3	27	77	41	7	18	1	41	17	1	6	21	21.19
(14 Lines)	FD	1548	6630	10336	801	9061	37199	11985	1800	7148	149	17634	6420	2543	3551	8343.21	9665.22
	AFD	309.6	288.26	382.81	267	335.59	483.1	292.32	257.14	397.11	149	430.1	377.65	2543	591.83	507.47	595.73
	R	3	7	9	2	8	8	11	5	9	1	12	8	1	3	6.21	3.68

Note. Participants with accurate answers are highlighted in green.

Table 4.16: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task B-1-B (repositioned)

	Metric	Treatment														
PID / Block		**P02**	**P03**	**P04**	**P08**	**P09**	**P11**	**P16**	**P18**	**P19**	**P20**	**P23**	**P24**	**P28**	**Mean**	**Std. Dev.**
Block 1	FC	1	0	4	0	8	2	3	3	0	5	14	0	2	3.23	4
(9 Lines)	FD	400	0	1882	0	2464	916	1649	1132	0	2118	9732	0	516	1600.69	2594.35
	AFD	400	0	470.5	0	308	458	549.67	377.33	0	423.6	695.14	0	258	303.1	235.49
	R	1	0	3	0	4	2	2	1	0	2	4	0	2	1.62	1.45
Block 2	FC	271	281	206	242	301	640	275	225	255	363	105	97	440	284.69	140.55
(20 Lines)	FD	84118	127038	116079	120995	141332	272683	140565	77787	145464	162946	54059	84824	180571	131420.08	
	AFD	310.4	452.09	563.49	499.98	469.54	426.07	511.15	345.72	570.45	448.89	514.85	874.47	410.39	492.11	137.79
	R	61	23	20	42	32	52	35	29	31	42	12	13	43	33.46	14.55
Block 3	FC	45	0	83	79	66	166	53	56	72	93	8	41	139	69.31	46.07
(10 Lines)	FD	11554	0	42336	31661	27843	81987	20436	26880	36064	34427	2669	32006	52728	30814.69	21445.68
	AFD	256.76	0	510.07	400.77	421.86	493.9	385.58	480	500.89	370.18	333.63	780.63	379.34	408.74	175.2
	R	19	0	11	5	12	8	9	6	6	9	4	3	10	7.85	4.78
Block 4	FC	79	3	46	7	42	5	10	11	36	24	9	1	23	22.77	22.67
(12 Lines)	FD	24612	800	24374	2132	19299	1583	2585	2668	23498	10614	2220	1806	7399	9506.92	9774.85
	AFD	311.54	266.67	529.87	304.57	459.5	316.6	258.5	242.55	652.72	442.25	246.67	1806	321.7	473.78	419.15
	R	24	1	10	6	12	3	5	4	8	6	5	1	12	7.46	6.15
Block 5	FC	4	0	31	0	23	2	5	3	0	4	5	1	10	6.77	9.53
(13 Lines)	FD	1617	0	13198	0	12112	817	3548	3033	0	1086	1816	300	2665	3091.69	4409.89
	AFD	404.25	0	425.74	0	526.61	408.5	709.6	1011	0	271.5	363.2	300	266.5	360.53	287.43
	R	2	0	4	0	3	2	4	2	0	2	3	1	4	2.08	1.5
Block 6	FC	4	0	7	3	19	3	38	7	0	6	3	0	7	7.46	10.44
(12 Lines)	FD	935	0	1952	583	8885	999	22274	2197	0	2069	1099	0	3016	3385.31	6134.09
	AFD	233.75	0	278.86	194.33	467.63	333	586.16	313.86	0	344.83	366.33	0	430.86	273.05	185.15
	R	3	0	4	2	5	3	18	4	0	4	3	0	4	3.85	4.58
Block 7	FC	7	0	5	0	11	7	4	3	0	5	7	0	2	3.92	3.5
(14 Lines)	FD	2184	0	1817	0	3516	2180	1367	867	0	3646	1549	0	1515	1433.92	1260.44
	AFD	312	0	363.4	0	319.64	311.43	341.75	289	0	729.2	221.29	0	757.5	280.4	251.87
	R	2	0	1	0	4	2	2	2	0	3	2	0	2	1.54	1.27

Note. Participants with accurate answers are highlighted in green.



(a) Fixation Count in Block 1

(b) Fixation Duration in Block 1

(c) Revisits of Block 1

(d) Revisits of Block 5

(e) Fixation Count of Block 7

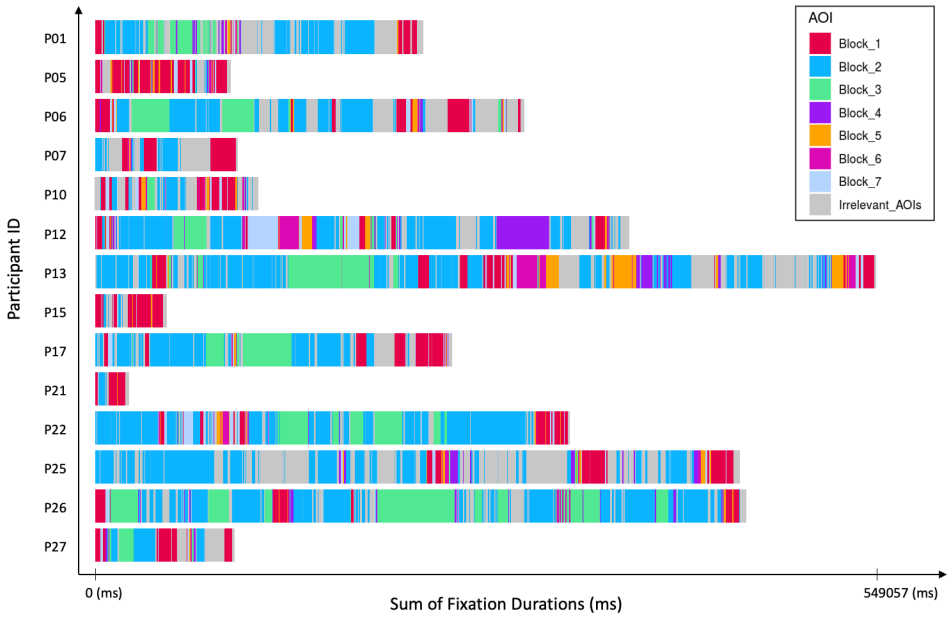
(f) Fixation Duration of Block 7

(g) Revisits of Block 7

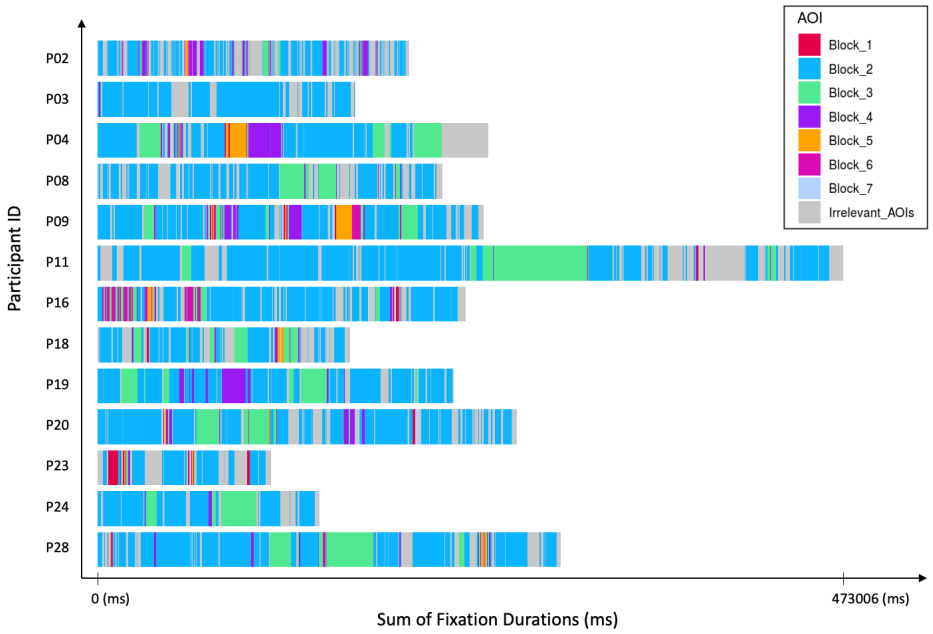
Figure 4.9: Box plots of B-1 metrics with significant differences in two groups

Figure 4.10 shows the scarf plot of the fixations on the relevant and irrelevant blocks for task B-1. As evident in the scarf plot and confirmed by the statistical tests, Block 1 which contains the assertion line in B-1-A is frequently revisited with more frequent and longer fixations. We can observe that most of the participants have focused on Blocks 2 and 3 while working on task B-1-B, and have paid less attention to the other blocks. We can also observe that the participants mostly looked at relevant blocks in the code.

After exploring the fixation based metrics, we ran the Mann-Whitney U test to compared the accuracy of the answers between the non-treatment and treatment



(a) Scarfplot of B-1-A



(b) Scarfplot of B-1-B

Figure 4.10: Scarfplots of Task B-1

Table 4.17: Statistical tests to compare metrics from B-2 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration)

Block	**Metric**	**Test**	**Statistic**	**df**	**p**	**Effect Size**
Block 1	FC	Mann-Whitney	182		<.001*	1
(10 Lines)	FD	Mann-Whitney	181		<.001*	0.989
	AFD	Mann-Whitney	158		0.001*	0.736
	Revisits	Mann-Whitney	180		<.001*	0.978
Block 2	FC	Mann-Whitney	174.5		<.001*	0.918
(9 Lines)	FD	Mann-Whitney	176		<.001*	0.934
	AFD	Mann-Whitney	176		<.001*	0.934
	Revisits	Mann-Whitney	176		<.001*	0.934
Block 3	FC	Mann-Whitney	175		<.001*	0.923
(8 Lines)	FD	Mann-Whitney	175		<.001*	0.923
	AFD	Mann-Whitney	127		0.084	0.396
	Revisits	Mann-Whitney	173.5		<.001*	0.907
Block 4	FC	Mann-Whitney	178		<.001*	0.956
(8 Lines)	FD	Mann-Whitney	177		<.001*	0.945
	AFD	Mann-Whitney	159		<.001*	0.747
	Revisits	Mann-Whitney	172.5		<.001*	0.896
Block 5	FC	Mann-Whitney	48		0.038*	-0.473
(13 Lines)	FD	Mann-Whitney	51	25	0.054	-0.44
	AFD	Student	-0.046		0.964	-0.018
	Revisits	Mann-Whitney	112		0.319	0.231

groups. The test suggested no significant difference between the accuracy of the two groups ($p = 0.595$, ES = -0.110).

Results for B-2: Repositioning of a Single Alarm

Tables 4.18 and 4.19 present the fixation metrics and their mean and standard deviation for the five relevant blocks of task B-2 in the two non-treatment and repositioned groups. Thirteen participants answered task B-2-A (non-treatment) and 14 participants answered the task B-2-B (repositioned). For our analysis, we first looked at the differences between sum of fixation count and sum of fixation duration on the entire file for the task, to see whether the repositioning treatment has had an effect on these metrics. The Mann-Whitney U test shows that despite the fixation

count for the non-treatment group was in average higher than the repositioned group, no significant differences between the number of fixations for the entire file ($p = 0.054$, ES = 0.440) were found. The same goes for the duration, and despite the higher average fixation duration on the repositioned treatment, there was no significant differences between the two groups based on the Mann-Whitney U test ($p = 0.061$, ES = 0.429). The repositioning treatment moved the assertion in the task from Block 1 to Block 5. This repositioning treatment draws attention to Block 5 which is the most relevant block in the code for answering the question. We looked at the differences of the metrics between each block.

Table 4.17 shows the results of the statistical tests to determine the differences between the fixation metrics on non-treatment and repositioned tasks. The results show significant differences in Block 1 Fixation Count ($p = < 0.001$, ES = 1), Fixation Duration ($p = < 0.001$, ES = 0.989), Average Fixation Duration ($p = 0.001$, ES = 0.736), Revisits ($p = < 0.001$, ES = 0.978), Block 2 Fixation Count ($p = < 0.001$, ES = 0.918), Fixation Duration ($p = < 0.001$, ES = 0.934), Average Fixation Duration ($p = < 0.001$, ES = 0.934), Revisits ($p = < 0.001$, ES = 0.934), Block 3 Fixation Count ($p = < 0.001$, ES = 0.923), Fixation Duration ($p = < 0.001$, ES = 0.923), Revisits ($p = < 0.001$, ES = 0.907), Block 4 Fixation Count ($p = < 0.001$, ES = 0.956), Fixation Duration ($p = < 0.001$, ES = 0.945), Average Fixation Duration ($p = < 0.001$, ES = 0.747), Revisits ($p = < 0.001$, ES = 0.896), and Block 5 Fixation Count ($p = 0.038$, ES = -0.473). The box plots representing the blocks with significant differences in metrics are shown in Figure 4.11

Table 4.18: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task B-2-A (non-treatment)

	Metrics	**Non-treatment**														
PID / **Block**		**P02**	**P03**	**P04**	**P08**	**P09**	**P11**	**P16**	**P18**	**P19**	**P20**	**P23**	**P24**	**P28**	**Mean**	**Std. Dev.**
Block 1	FC	50	61	76	273	81	211	57	68	73	62	188	110	206	116.62	75.14
(10 Lines)	FD	13031	29410	32808	125344	34727	81745	28378	24581	42130	25241	79887	60050	74410	50134	32122.65
	AFD	260.62	482.13	431.68	459.14	428.73	387.42	497.86	361.49	577.12	407.11	424.93	545.91	361.21	432.72	83.49
	R	17	10	25	49	18	38	19	20	9	16	31	13	60	25	15.51
Block 2	FC	24	2	56	35	44	61	55	32	46	13	68	49	118	46.39	28.74
(9 Lines)	FD	8167	700	24502	17698	22144	24417	31497	10083	24200	4818	27480	31316	54022	21618.77	13988.92
	AFD	340.29	350	437.54	505.66	503.27	400.28	572.67	315.09	526.09	370.62	404.12	639.1	457.81	447.89	97.39
	R	8	1	11	9	13	10	11	7	5	5	13	12	29	10.31	6.64
Block 3	FC	26	2	12	16	38	60	30	18	19	27	50	49	83	33.08	22.37
(8 Lines)	FD	7048	334	5649	5214	14284	19427	16740	7335	12380	8797	18188	26686	26645	12979	8256.31
	AFD	271.08	167	470.75	325.88	375.89	323.78	558	407.5	651.58	325.81	363.76	544.61	321.02	392.82	132.1
	R	10	1	6	10	15	12	15	10	6	9	13	19	39	12.69	9.15
Block 4	FC	16	5	39	17	21	67	18	35	8	19	21	71	74	31.62	24.08
(8 Lines)	FD	3729	1334	15368	6297	8469	23628	6544	10730	2168	8603	8430	30920	27148	11797.54	9630.46
	AFD	233.06	266.8	394.05	370.41	403.29	352.66	363.56	306.57	271	452.79	401.43	435.49	366.86	355.23	67.43
	R	5	1	8	5	7	10	8	9	2	9	6	22	27	9.15	7.38
Block 5	FC	44	186	287	34	98	602	159	114	113	204	45	83	271	172.31	153.06
(13 Lines)	FD	14097	86963	131743	13063	48344	238613	70118	40542	54230	86239	17074	56774	80780	72198.46	60398.6
	AFD	320.39	467.54	459.03	384.21	493.31	396.37	440.99	355.63	479.91	422.74	379.42	684.02	298.08	429.36	97.55
	R	18	17	27	17	15	45	32	24	18	32	4	21	59	25.31	14.26

Note. Participants with accurate answers are highlighted in green.

Table 4.19: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task B-2-B (repositioned)

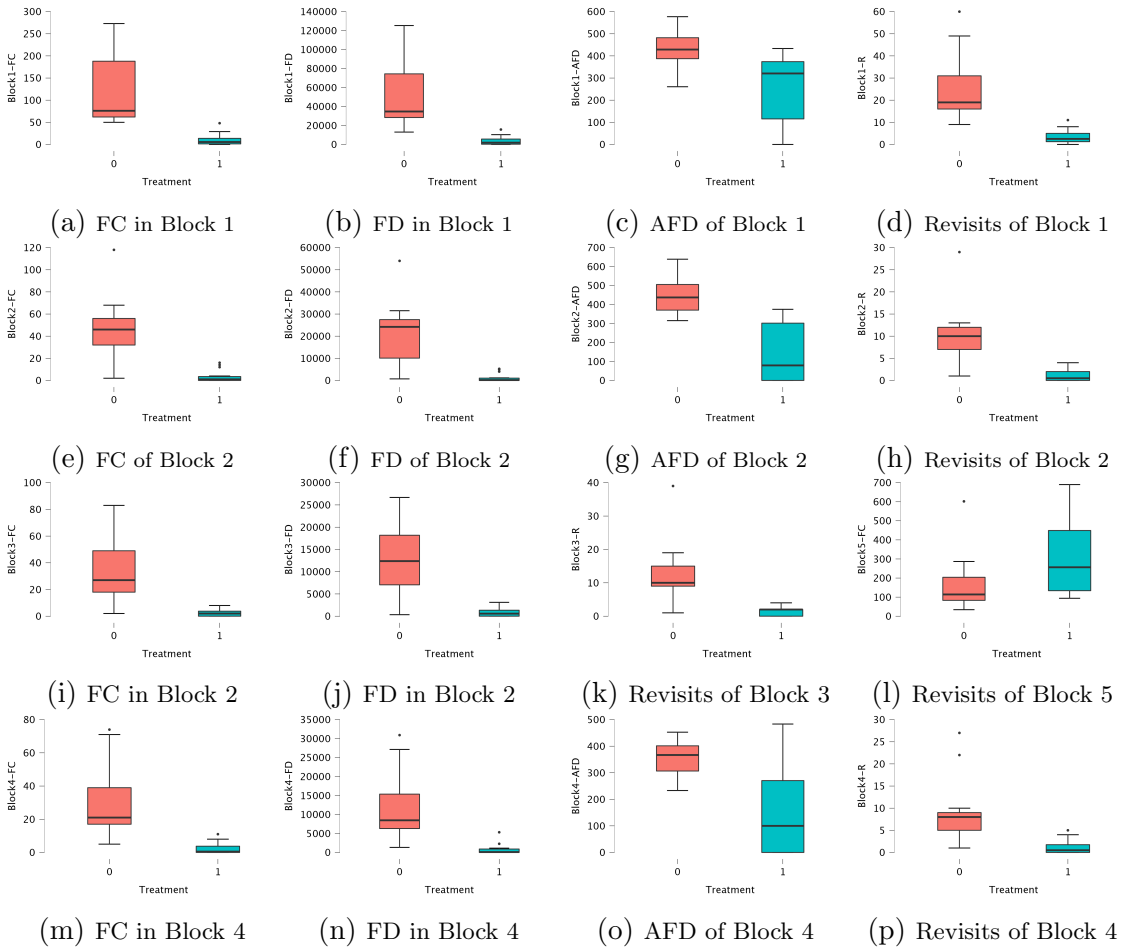
	Metrics	Treatment														
PID / **Block**		**P01**	**P05**	**P06**	**P07**	**P10**	**P12**	**P13**	**P15**	**P17**	**P21**	**P22**	**P25**	**P26**	**P27**	**Mean**	**Std. Dev.**
Block 1	FC	0	4	13	29	6	14	0	2	1	48	5	22	0	6	10.71	13.88
(10 Lines)	FD	0	851	4932	10296	1885	5917	0	716	83	15708	1917	6635	0	2600	3681.43	4654.18
	AFD	0	212.75	379.38	355.03	314.17	422.64	0	358	83	327.25	383.4	301.59	0	433.33	255.04	164.04
	R	0	2	5	11	3	5	0	2	1	5	2	8	0	5	3.5	3.23
Block 2	FC	0	0	2	14	4	16	0	2	0	12	0	2	0	0	3.71	5.76
(9 Lines)	FD	0	0	632	5250	1184	4863	0	317	0	3997	0	482	0	0	1194.64	1948.19
	AFD	0	0	316	375	296	303.94	0	158.5	0	333.08	0	241	0	0	144.54	157.42
	R	0	0	2	4	3	3	0	1	0	2	0	1	0	0	1.14	1.41
Block 3	FC	0	0	3	7	2	8	0	6	0	4	2	2	0	2	2.57	2.74
(8 Lines)	FD	0	0	1399	3117	299	2734	0	2634	0	1084	833	315	0	1167	970.14	1120.73
	AFD	0	0	466.33	445.29	149.5	341.75	0	439	0	271	416.5	157.5	0	583.5	233.6	214.09
	R	0	0	2	4	2	2	0	3	0	2	1	2	0	2	1.43	1.28
Block 4	FC	0	0	5	11	4	3	0	1	0	1	0	8	0	0	2.36	3.5
(8 Lines)	FD	0	0	1132	5318	851	917	0	200	0	317	0	2281	0	0	786.86	1462.43
	AFD	0	0	226.4	483.45	212.75	305.67	0	200	0	317	0	285.13	0	0	145.03	164.32
	R	0	0	2	5	1	2	0	1	0	1	0	4	0	0	1.14	1.61
Block 5	FC	101	115	479	482	94	279	360	234	315	130	223	690	514	145	297.21	185.37
(13 Lines)	FD	50516	43222	215555	254166	34880	130340	137934	97716	124618	43245	88322	288548	251622	73455	131009.93	87329.18
	AFD	500.16	375.84	450.01	527.32	371.06	467.17	383.15	417.59	395.61	332.65	396.06	418.19	489.54	506.59	430.78	59.67
	R	10	26	18	58	10	17	24	16	25	11	15	68	13	20	23.64	17.59

Note. Participants with accurate answers are highlighted in green.

Figure 4.12 shows the scarf plots from tasks B-1-A (non-treatment) and B-1-B (repositioned treatment). The scarf plots show that the participants paid attention to all the relevant blocks and went back and forth between them in B-2-A, but their visual attention was mostly on Block 5 when they worked on B-2-B, the treatment task.

After exploring the fixation based metrics, we ran the Mann-Whitney U test to compare the accuracy of the answers between the non-treatment and treatment groups. Due to the fact that only two participants answered the task inaccurately, we were not able to run statistical tests to look at the differences.

Our findings from the two B group tasks give us evidence to test the $REHV$ and $REHA$ hypotheses. The evidence from comparing the visual effort on blocks shows us that there are significant differences between the visual effort on blocks for different treatments. Therefore, we can accept the alternate hypothesis, $REHV_A$. However, for hypothesis $REHA$, the evidence did not show any effects from repositioning on the accuracy of both tasks, so we cannot reject the null hypothesis $REHA_0$.



(a) FC in Block 1 (b) FD in Block 1 (c) AFD of Block 1 (d) Revisits of Block 1

(e) FC of Block 2 (f) FD of Block 2 (g) AFD of Block 2 (h) Revisits of Block 2

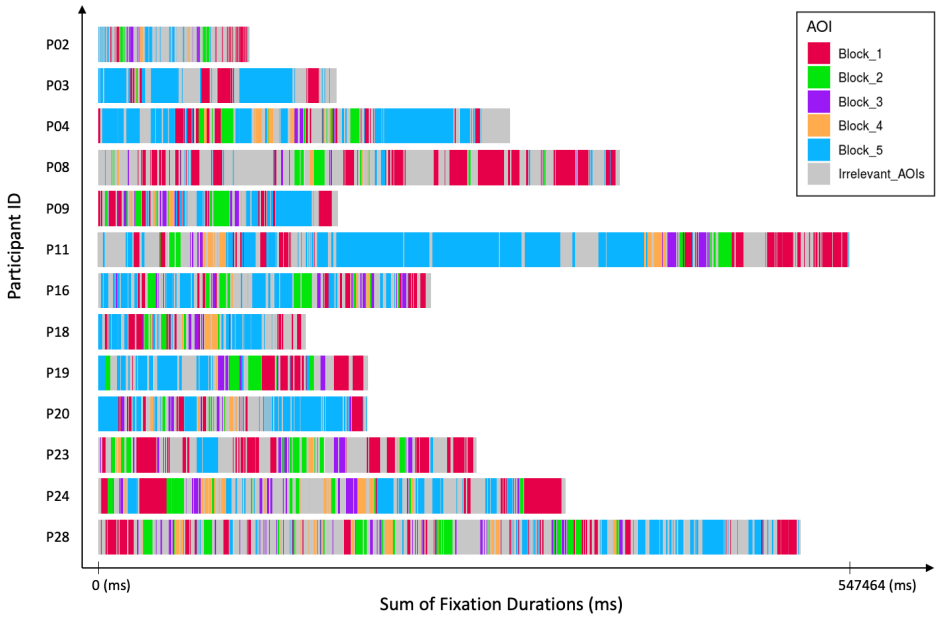
(i) FC in Block 2 (j) FD in Block 2 (k) Revisits of Block 3 (l) Revisits of Block 5

(m) FC in Block 4 (n) FD in Block 4 (o) AFD of Block 4 (p) Revisits of Block 4

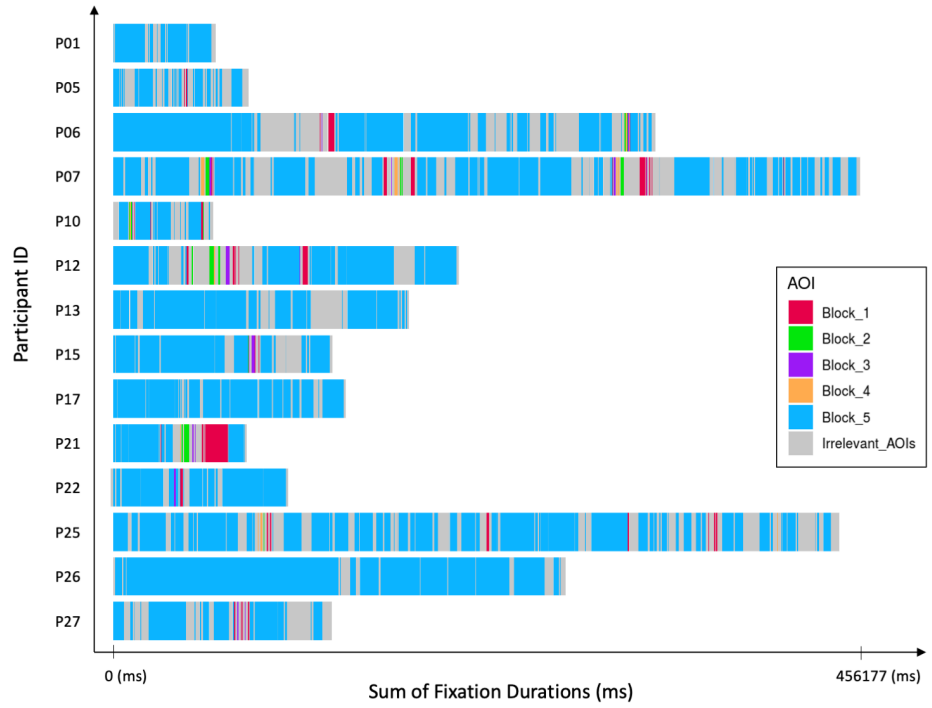
Figure 4.11: Box plots of B-2 metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration) with significant differences in two groups

4.5.5 RQ3: Effect of Repositioning and Merging Multiple Alarms

To answer RQ3 and test hypotheses $MEHV$ and $MEHA$, we investigated the fixation metrics from group C : Merging of Similar Alarms. Of the 27 study participants, 26 completed the C tasks, due to a technical issue that prevented us from saving participant P11's data. We collected the answer to the question from all participants. There are two tasks in this category, namely C-1 and C-2. The participants received one of the tasks without any treatment (i.e. a number of alarms in their original location) and the other task with the merging treatment (i.e. all the alarms merged into one alarm and repositioned). Similar to the B group tasks, the tasks without



(a) Scarfplot of B-2-A



(b) Scarfplot of B-2-B

Figure 4.12: Scarfplots of Task B-2

treatment are specified with A at the end of their name (e.g. C-1-A) and the tasks with the merging treatment are specified with B at the end of their name (e.g. C-2-B). For Task Group C tasks, the following fixation metrics from the participants are generated: Fixation Count (FC), Fixation Duration (FD), Average Fixation Duration (AFD), and Revisits. Each task in the C group asked the participants to inspect and evaluate multiple assertions over one or multiple files. Our analysis for task C-1 is very similar to the Group B tasks analysis, as the participant only sees one file for the task. For task C-2, the participant is presented with a folder that contains 38 files, and the assertions for the task are in multiple files. Thus, we perform a file level analysis for this task as well as block level analysis.

To test the hypothesis $MEHV$ and to explore the effects of merging multiple alarms on visual effort, we look at the fixation metrics on the files and then relevant blocks of them, and we compare them between non-treatment and treatment groups. Scarf plots are also generated for the tasks to visualize the fixation patterns of the participants in each group. To test hypothesis $MEHA$ and understand the effect of merging on the accuracy, we compared the accuracy of the participant's answers between the two groups. The answer to each question about an assertion is either "Yes" or "No" and we denote the correctness of the answers as 1 or 0 in our analysis.

Table 4.20: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task C-1-A (non-treatment)

	Metric	Non-treatment													
PID / **Block**		**P02**	**P06**	**P10**	**P13**	**P15**	**P16**	**P18**	**P19**	**P21**	**P22**	**P25**	**P28**	**Mean**	**Std. Dev**
Block 1	FC	36	72	15	42	72	37	24	37	14	78	133	36	49.67	33.89
(10 Lines)	FD	11300	26297	6150	14045	24243	14213	5985	14661	5046	28209	47914	12948	17584.25	12317.56
	AFD	313.89	365.24	410	334.4	336.71	384.14	249.38	396.24	360.43	361.65	360.26	359.67	352.67	41.93
	R	13	12	4	10	8	14	4	7	2	10	30	14	10.67	7.32
Block 2	FC	12	26	4	13	15	19	4	5	10	4	51	31	16.17	14.06
(7 Lines)	FD	3017	8500	1167	4463	6403	6500	3333	2467	2018	1001	15232	11221	5443.5	4374.46
	AFD	251.42	326.92	291.75	343.31	426.87	342.11	833.25	493.4	201.8	250.25	298.67	361.97	368.48	166.34
	R	5	8	2	4	6	10	3	4	3	3	15	15	6.5	4.58
Block 3	FC	4	25	2	11	7	5	9	12	4	2	33	28	11.83	10.78
(7 Lines)	FD	2115	7443	899	2817	3099	2697	3514	3582	934	234	10702	10117	4012.75	3506.61
	AFD	528.75	297.72	449.5	256.09	442.71	539.4	390.44	298.5	233.5	117	324.3	361.32	353.27	124.57
	R	3	10	2	3	3	4	4	3	1	1	16	13	5.25	4.94
Block 4	FC	11	20	3	13	7	3	3	1	1	1	25	21	9.08	8.76
(11 Lines)	FD	2470	6749	600	4165	3083	3266	633	1166	317	949	8919	7365	3306.83	2936.61
	AFD	224.55	337.45	200	320.38	440.43	1088.67	211	1166	317	949	356.76	350.71	496.83	354.13
	R	7	7	3	5	5	3	1	1	1	1	7	10	4.25	3.05
Block 5	FC	8	23	9	3	4	1	7	2	0	8	15	6	7.17	6.48
(11 Lines)	FD	2267	10153	3433	1816	949	517	1449	1466	0	3265	3614	2766	2641.25	2635.08
	AFD	283.38	441.43	381.44	605.33	237.25	517	207	733	0	408.13	240.93	461	376.33	197.37
	R	5	11	2	1	2	1	4	2	0	1	6	6	3.42	3.15
Block 6	FC	76	299	92	237	287	244	71	154	12	206	472	267	201.42	128.03
(41 Lines)	FD	20935	120320	25353	90196	119881	109034	30874	85892	4150	82456	155807	98275	78597.75	47580.09
	AFD	275.46	402.41	275.58	380.57	417.7	446.86	434.85	557.74	345.83	400.27	330.1	368.07	386.29	77.79
	R	13	29	17	22	17	18	6	8	7	27	42	35	20.08	11.37
Block 7	FC	3	206	0	0	165	45	78	0	0	0	7	142	53.83	75.81
(7 Lines)	FD	1517	90986	0	0	119667	24641	30422	0	0	0	1978	56872	27173.58	41022.29
	AFD	505.67	441.68	0	0	725.25	547.58	390.03	0	0	0	282.57	400.51	274.44	263.75
	R	2	21	0	0	8	19	17	0	0	0	5	28	8.33	10.16

Note. Participants with accurate answers are highlighted in green.

Table 4.21: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task C-1-B (merging treatment)

	Metric	Treatment															
PID / Block		**P01**	**P03**	**P04**	**P05**	**P07**	**P08**	**P09**	**P12**	**P17**	**P20**	**P23**	**P24**	**P26**	**P27**	**Mean**	**Std. Dev.**
Block 1	FC	1	0	3	0	0	12	13	5	0	2	0	1	0	3	2.86	4.37
(10 Lines)	FD	217	0	1251	0	0	4303	7133	1800	0	1350	0	743	0	1149	1281.86	2055.73
	AFD	217	0	417	0	0	358.58	548.69	360	0	675	0	743	0	383	264.45	271.11
	R	1	0	1	0	0	4	4	4	0	2	0	1	0	1	1.29	1.59
Block 2	FC	0	0	2	0	0	6	7	5	3	5	0	0	0	0	2	2.66
(7 Lines)	FD	0	0	1284	0	0	2334	2328	1601	1481	2549	0	0	0	0	826.93	1045
	AFD	0	0	642	0	0	389	332.57	320.2	493.67	509.8	0	0	0	0	191.95	242.49
	R	0	0	2	0	0	3	4	4	3	4	0	0	0	0	1.43	1.79
Block 3	FC	2	0	1	0	0	4	3	8	1	1	0	1	0	1	1.57	2.21
(7 Lines)	FD	917	0	250	0	0	2565	1200	2466	451	234	0	374	0	599	646.86	872.78
	AFD	458.5	0	250	0	0	641.25	400	308.25	451	234	0	374	0	599	265.43	233.61
	R	2	0	1	0	0	3	2	5	1	1	0	1	0	1	1.21	1.42
Block 4	FC	1	0	1	0	0	4	3	12	0	5	0	3	0	0	2.07	3.34
(11 Lines)	FD	267	0	816	0	0	2019	1084	4566	0	2182	0	978	0	0	850.86	1313.22
	AFD	267	0	816	0	0	504.75	361.33	380.5	0	436.4	0	326	0	0	220.86	260.34
	R	1	0	1	0	0	4	2	4	0	4	0	2	0	0	1.29	1.64
Block 5	FC	1	0	4	0	0	9	4	2	1	1	0	3	0	3	2	2.51
(11 Lines)	FD	734	0	1533	0	0	3150	1951	467	367	350	0	1922	0	867	810.07	974.75
	AFD	734	0	383.25	0	0	350	487.75	233.5	367	350	0	640.67	0	289	273.94	248.24
	R	1	0	4	0	0	4	2	2	1	1	0	3	0	1	1.36	1.45
Block 6	FC	103	98	178	119	226	373	120	154	363	325	207	222	362	77	209.07	107.03
(41 Lines)	FD	63085	46842	85400	48577	119374	201882	54508	64685	154794	162136	83121	170880	204447	46131	107561.57	59679.04
	AFD	612.48	477.98	479.78	408.21	528.2	541.24	454.23	420.03	426.43	498.88	401.55	769.73	564.77	599.1	513.04	100.94
	R	3	4	7	12	13	23	17	13	19	15	11	8	7	10	11.57	5.68
Block 7	FC	21	5	93	0	67	82	68	1	156	215	62	66	0	26	61.57	62.83
(7 Lines)	FD	10450	1183	57327	0	35466	44948	30298	1084	76319	101087	30637	51295	0	16146	32588.57	31132.16
	AFD	497.62	236.6	616.42	0	529.34	548.15	445.56	1084	489.22	470.17	494.15	777.2	0	621	486.39	280.12
	R	7	1	18	0	7	9	9	1	18	23	4	13	0	4	8.14	7.4

Note. Participants with accurate answers are highlighted in green.

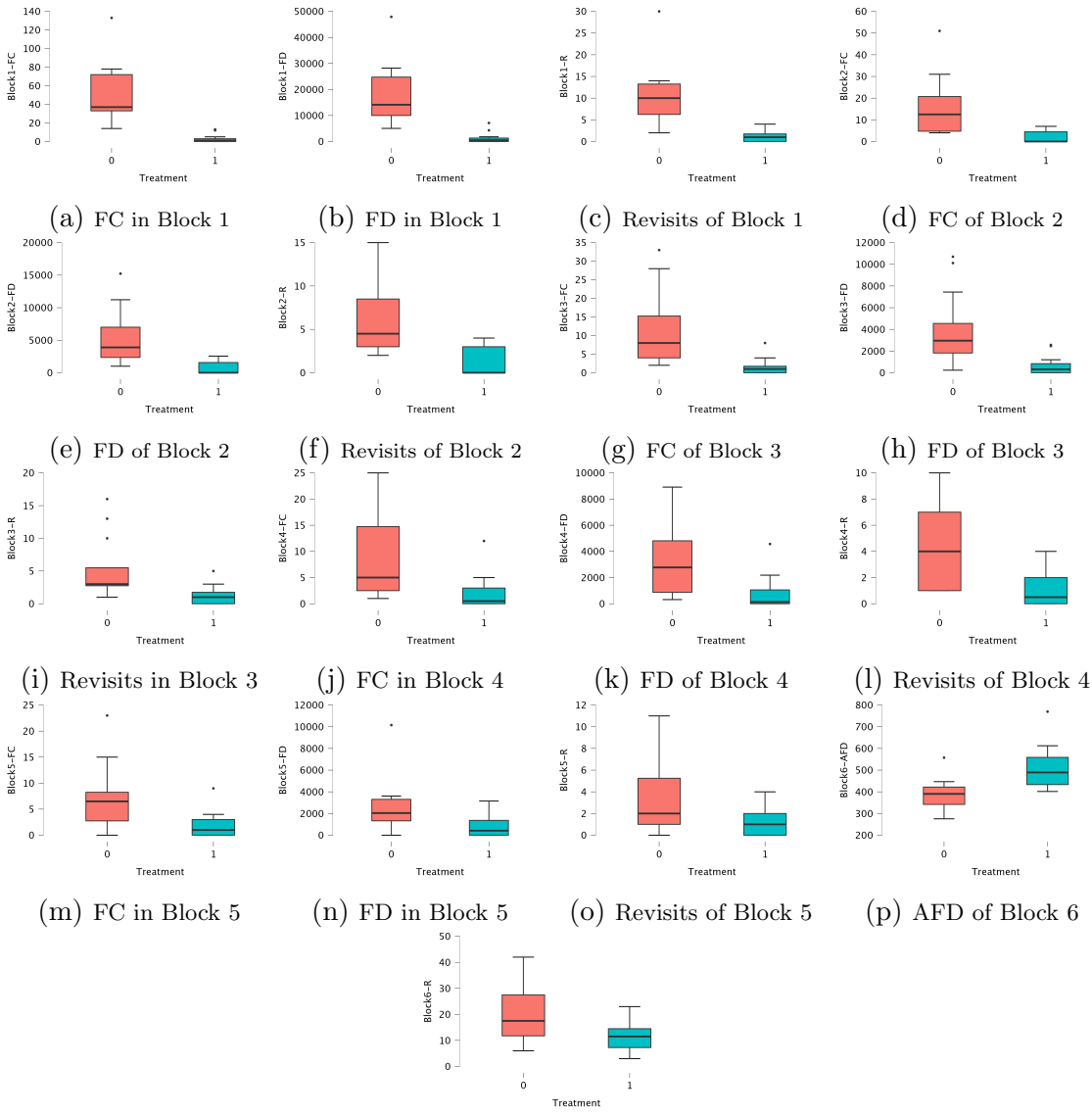
Results for C-1: Merging and Repositioning of Multiple Alarms.

Tables 4.20 and 4.21 show the fixation metrics, their mean and standard deviation for the 7 relevant blocks of task C-1. Out of the 26 participants, 12 participants completed the C-1-A task and 14 participants completed the C-1-B task. For our analysis, we first looked at the differences between the sum of fixation count and fixation duration over the entire file between the two group, to examine whether the merging treatment had any effects on these metrics. Even though on average, the fixation count and duration for C-1-A were higher than C-1-B, the Mann-Whitney U test showed no significant differences between fixation count ($p = 0.527$, ES $= 0.155$) and fixation duration ($p = 0.860$, ES $= 0.048$).

Multiple assertions are specified in task C-1-A, on Blocks 1, 2, 3, 4, and 5, while the C-1-B task only contains one merged assertion on Block 6. The merging and repositioning is done to draw the attention of the developer to Blocks 6 and 7, in which they can find the answer to the question. We then compared the fixation metrics between the 7 Blocks for the two treatments. Table 4.22 shows the statistical tests performed for the comparison. The tests show significant differences between Block 1 Fixation Count ($p = < 0.001$, ES $= 1$), Fixation Duration ($p = < 0.001$, ES $= 0.964$), and Revisits ($p = < 0.001$, ES $= 0.923$), Block 2 Fixation Count ($p = < 0.001$, ES $= 0.821$), Fixation Duration ($p = < 0.001$, ES $= 0.81$), and Revisits ($p = < 0.001$, ES $= 0.756$), Block 3 Fixation Count ($p = < 0.001$, ES $= 0.857$), Fixation Duration ($p = < 0.001$, ES $= 0.792$), and Revisits ($p = 0.002$, ES $= 0.714$), Block 4 Fixation Count ($p = 0.007$, ES $= 0.619$), Fixation Duration ($p = 0.005$, ES $= 0.643$), and Revisits ($p = 0.005$, ES $= 0.643$), Block 5 Fixation Count ($p = 0.012$, ES $= 0.583$), Fixation Duration ($p = 0.011$, ES $= 0.589$), and Revisits ($p = 0.046$, ES $= 0.458$),

Table 4.22: Statistical tests to compare metrics from C-1 blocks (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration

Block	Metric	Test	Statistic	df	p	Effect Size
Block 1	FC	Mann-Whitney	168		<.001*	1
(10 Lines)	FD	Mann-Whitney	165		<.001*	0.964
	AFD	Mann-Whitney	102		0.365	0.214
	Revisits	Mann-Whitney	161.5		<.001*	0.923
Block 2	FC	Mann-Whitney	153		<.001*	0.821
(7 Lines)	FD	Mann-Whitney	152		<.001*	0.81
	AFD	Mann-Whitney	115		0.111	0.369
	Revisits	Mann-Whitney	147.5		<.001*	0.756
Block 3	FC	Mann-Whitney	156		<.001*	0.857
(7 Lines)	FD	Mann-Whitney	150.5		<.001*	0.792
	AFD	Welch's t-test	1.219	20.402	0.237	0.469
	Revisits	Mann-Whitney	144		0.002*	0.714
Block 4	FC	Mann-Whitney	136		0.007*	0.619
(11 Lines)	FD	Mann-Whitney	138		0.005*	0.643
	AFD	Mann-Whitney	118		0.082	0.405
	Revisits	Mann-Whitney	138		0.005*	0.643
Block 5	FC	Mann-Whitney	133		0.012*	0.583
(11 Lines)	FD	Mann-Whitney	133.5		0.011*	0.589
	AFD	Student's t-test	1.15	24	0.262	0.452
	Revisits	Mann-Whitney	122.5		0.046*	0.458
Block 6	FC	Student's t-test	-0.166	24	0.869	-0.065
(41 Lines)	FD	Mann-Whitney	66		0.374	-0.214
	AFD	Student's t-test	-3.538	24	0.002*	-1.392
	Revisits	Welch's t-test	2.353	15.603	0.032*	0.947
Block 7	FC	Mann-Whitney	66		0.363	-0.214
(7 Lines)	FD	Mann-Whitney	64		0.311	-0.238
	AFD	Mann-Whitney	46		0.051	-0.452
	Revisits	Mann-Whitney	74		0.621	-0.119



(a) FC in Block 1 (b) FD in Block 1 (c) Revisits of Block 1 (d) FC of Block 2

(e) FD of Block 2 (f) Revisits of Block 2 (g) FC of Block 3 (h) FD of Block 3

(i) Revisits in Block 3 (j) FC in Block 4 (k) FD of Block 4 (l) Revisits of Block 4

(m) FC in Block 5 (n) FD in Block 5 (o) Revisits of Block 5 (p) AFD of Block 6

(q) Revisits of Block 6

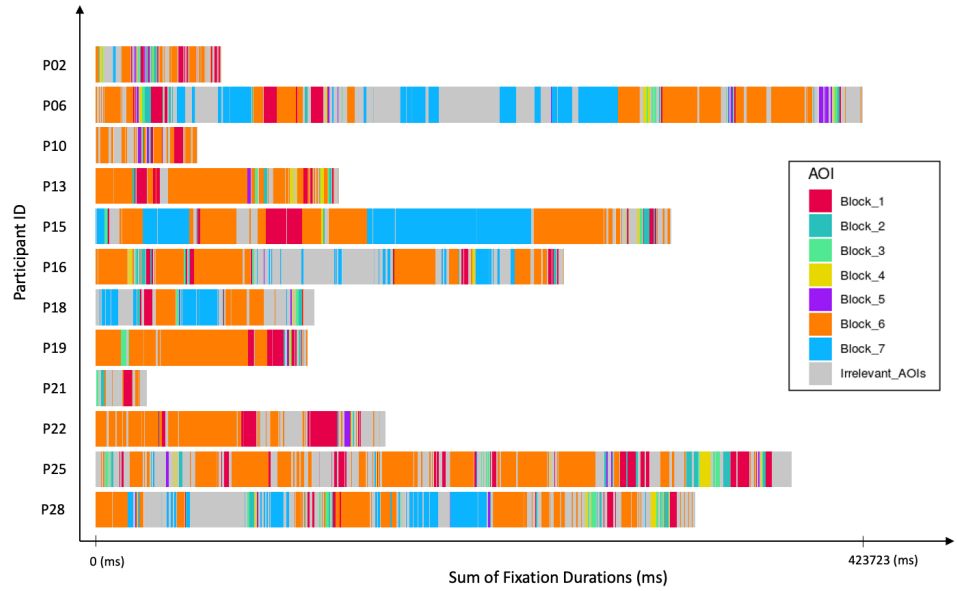
Figure 4.13: Box plots of C-1 metrics with significant differences in two groups

Block 6 Average Fixation Duration ($p = 0.002$, ES = -1.392), and Revisits ($p = 0.032$, ES = 0.947). Figure 4.13 shows the box plots of the metrics with significant differences.

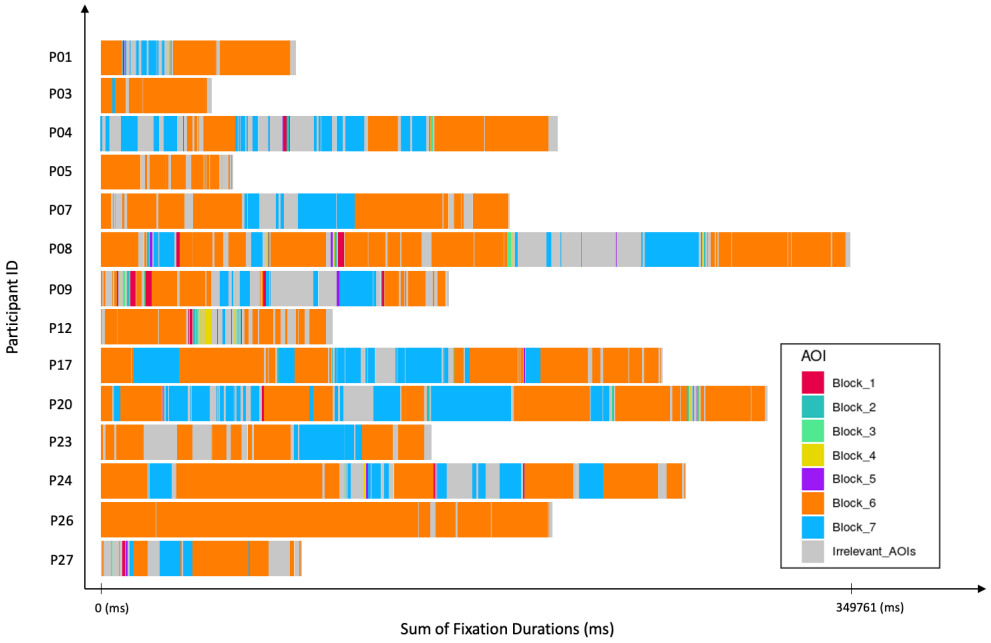
Figure 4.14 shows the scarf plots of tasks C-1-A and C-1-B. While both scarf plots show a lot of visual attention on Block 6 and 7, we can see that participants looked at the other blocks as well when they worked on the C-1-A task. The attention of the participants was mostly on Block 6 and Block 7 when working on the C-1-B task, and

the patterns show that participants went back and forth between these two blocks and they all finally focused their attention back on Block 6 before finishing the task.

To compare the accuracy of the answers between the non-treatment and merging treatment, we ran the Mann-Whitney U test on the scores of participants in two groups. The test did not show any significant differences ($p = 0.624$, ES $= 0.066$).



(a) Scarfplot of C-1-A



(b) Scarfplot of C-1-B

Figure 4.14: Scarf plots of Task C-1

Table 4.23: Comparison of metrics (FC = Fixation Count, FD = Fixation Duration) of files looked in Task C-2

Treatment Type	Participant ID	Number of Files Looked	FC on All Files	FC on Relevant Files	Percentage of Relevant Fixations	FD on All Files	FD on Relevant Files	Percentage of Relevant Fixations	Percentage of Relevant Files
Non-merged	P01	2	414	414	1	188870	188870	1	29%
	P03	4	343	306	89%	136915	126764	93%	29%
	P04	18	627	538	86%	304987	275394	90%	71%
	P05	2	276	276	1	120007	120007	1	29%
	P07	2	513	513	1	343747	343747	1	29%
	P08	7	822	759	92%	452689	423914	94%	57%
	P09	14	1385	1239	89%	603316	543331	90%	71%
	P12	5	549	466	85%	237801	206118	87%	29%
	P17	2	506	506	1	198288	198288	1	29%
	P20	8	1025	998	97%	515894	503652	98%	57%
	P23	16	1043	821	79%	577229	390348	68%	43%
	P24	3	591	591	1	502832	502832	1	43%
	P26	1	587	587	1	338952	338952	1	14%
	P27	7	441	389	88%	279654	256692	92%	29%
Merged	P02	3	954	954	1	323199	323199	1	1
	P06	6	754	578	77%	375728	309116	82%	33%
	P10	9	458	389	85%	200710	170241	85%	33%
	P13	9	1767	1286	73%	724216	528663	73%	33%
	P15	1	266	266	1	128617	128617	1	1
	P16	30	917	604	66%	470633	326788	69%	10%
	P18	6	859	801	93%	320183	296219	93%	50%
	P19	8	606	545	90%	303736	276808	91%	25%
	P21	1	194	194	1	64214	64214	1	1
	P22	8	894	591	66%	360921	237026	66%	25%
	P25	7	1654	1559	94%	645575	549137	85%	29%
	P28	10	1295	948	73%	492674	364059	74%	20%

Note. Participants with accurate answers are highlighted in green.

Table 4.24: Statistical tests for C-2 file metrics

Metric	**Test**	**W**	**df**	**p**	**Effect Size**
Number of Files Looked	Mann-Whitney	68.5		0.438	-0.185
Percentage of Relevant Files Looked	Mann-Whitney	84.5		1	0.006
Percentage of Relevant Fixation Duration	Mann-Whitney	119.5		0.066	0.423
Percentage of Relevant Fixation Count	Mann-Whitney	113.5		0.127	0.351

Table 4.25: Metrics (FC = Fixation Count, FD = Fixation Duration, AFD = Average Fixation Duration, R = Revisits) for relevant blocks of task C-2.

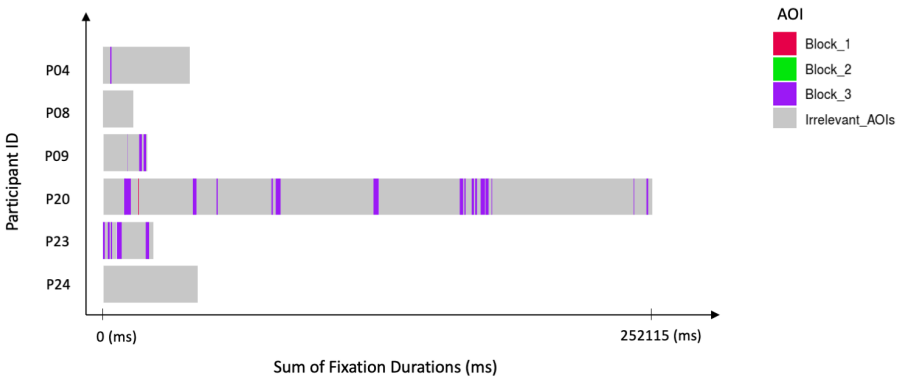
File	Block	Metric	Non-Treatment				Treatment										
			P04	**P09**	**P20**	**P23**	**P02**	**P06**	**P10**	**P13**	**P16**	**P18**	**P19**	**P21**	**P22**	**P25**	**P28**
	Block 1	FC	0	0	1	0	19	4	0	97	20	5	49	0	20	51	12
		FD	0	0	166	0	8198	2448	0	27690	11947	1118	16176	0	6131	18848	4043
		AFD	0	0	166	0	431.47	612	0	285.46	597.35	223.6	330.12	0	306.55	369.57	336.92
		R	0	0	1	0	10	1	0	31	8	3	19	0	6	25	6
	Block 2	FC	0	0	0	0	5	0	7	4	2	2	0	0	0	2	0
Archimedes.c		FD	0	0	0	0	1335	0	3482	750	768	434	0	0	0	262	0
		AFD	0	0	0	0	267	0	497.43	187.5	384	217	0	0	0	131	0
		R	0	0	0	0	3	0	3	2	2	2	0	0	0	1	0
	Block 3	FC	1	7	25	6	9	1	1	3	3	1	3	0	4	9	0
		FD	584	2284	18377	5851	8917	501	999	1749	900	716	3516	0	982	3842	0
		AFD	584	326.29	735.08	975.17	990.78	501	999	583	300	716	1172	0	245.5	426.89	0
		R	1	3	16	5	7	1	1	3	3	1	3	0	3	6	0
			P08	**P09**			**P02**	**P10**	**P16**	**P18**							
	Block 1	FC	27	60			24	8	8	23							
		FD	16798	21827			8098	3683	6233	7631							
		AFD	622.15	363.78			337.47	460.38	779.13	331.79							
readinputfile.h		R	9	19			14	4	7	13							
	Block 2	FC	3	126			52	10	44	0							
		FD	1600	62684			21241	6816	28016	0							
		AFD	533.33	497.5			408.48	681.6	636.73	0							
		R	2	25			12	5	5	0							
			P04				**P02**	**P06**	**P10**	**P13**	**P15**	**P16**	**P18**	**P19**	**P21**	**P22**	**P25**	**P28**
	Block 1	FC	0				24	93	32	236	94	46	49	63	91	119	241	190
		FD	0				8466	66465	15260	99180	61336	28762	21824	45303	31412	42745	99626	80944
		AFD	0				352.75	714.68	476.88	420.25	652.51	625.26	445.39	719.1	345.19	359.2	413.39	426.02
		R	0				15	32	10	58	20	22	24	25	12	36	53	42
	Block 2	FC	1				0	3	1	59	6	4	10	8	4	0	35	23
updating.h		FD	1167				0	1400	651	21169	2834	2148	3932	4067	969	0	13678	6698
		AFD	1167				0	466.67	651	358.8	472.33	537	393.2	508.38	242.25	0	390.8	291.22
		R	1				0	3	1	23	5	4	6	6	3	0	19	13
	Block 3	FC	2				16	24	5	13	14	6	10	9	1	7	6	19
		FD	332				4850	11321	2035	4232	7335	2350	3232	3753	167	3165	2510	7595
		AFD	166				303.13	471.71	407	325.54	523.93	391.67	323.2	417	167	452.14	418.33	399.74
		R	2				12	17	5	11	10	6	6	9	1	6	6	14

Results for C-2: Merging and Repositioning of Multiple Alarms

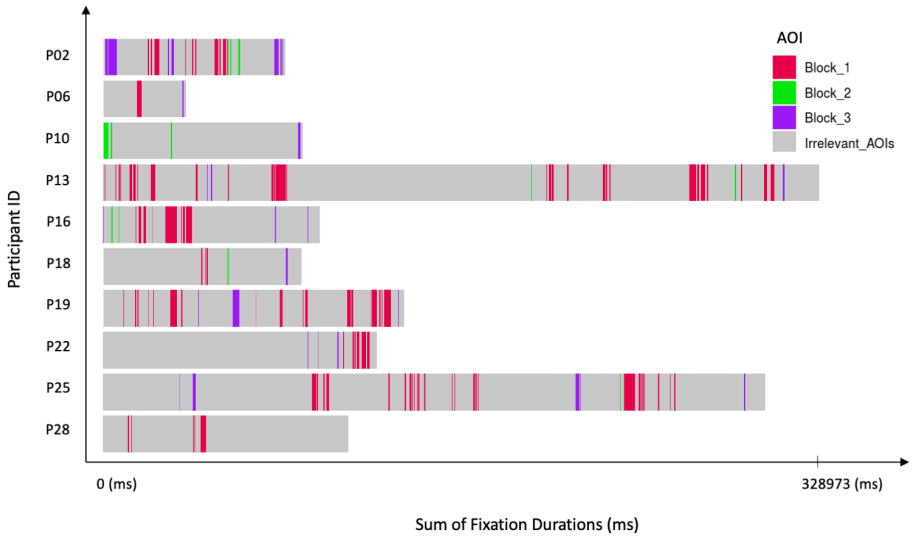
In task C-2, the participants are presented with a folder containing 38 files written in the C language. Fourteen participants worked on the non-treatment task (C-2-A), and 12 participants worked on the merged treatment task (C-2-B). In C-2-A, four assertions were presented on two different files (`Hole_bcs.h` and `HMEPbcs.h`), and in C-2-B, one assertion was presented on one file (`updating.h`). There were 7 relevant files for solving the C-2-A task. The merged treatment reduced the number of relevant files to 3 files, making three files common to solve the task for both treatments. The participants were first shown the files that included the assertions but were told that they could explore the project and look at any of the 38 files. Due to the fact that participants looked at many files while working on this task, to compare the non-treatment and merged tasks we first look at some file level metrics. We look at the number of files looked at in each task, the sum of fixation count on all files, the sum of fixation count on relevant files, and the percentage of relevant fixations (which are fixations that are on relevant files). We also look at the same information about fixation duration on the files. Table 4.23 shows the file metrics for both of the treatments. Table 4.24 shows the statistical tests for C-2 file metrics, to determine whether the treatment has had any effect on the number of files looked, and percentages of relevant files, fixation count, and fixation duration. We did not directly compare the number of fixations because the number of relevant files were different for the different treatments of the task. The tests did not indicate there were any significant differences between the file metrics.

Additionally, we looked at the fixation metrics on the blocks of the common relevant files for both the non-treatment and merged tasks. In order to solve the non-treatment task, participants had to follow a series of function calls from file to file, which would

eventually lead them to the last three files. In the treatment task, the assertion was placed in one of these three files, with the goal of reducing the number of file traversals and visual effort for answering the question. However, Table 4.25 shows us that the majority of participants from the non-treatment group did not fixate on the blocks containing relevant information for solving the task. This is also evident in scarf plots in Figures 4.15, 4.16, 4.17. The figures also show us that the participants looked at irrelevant AOIs frequently and for long times, indicating that it was difficult for them to find important information relevant to the tasks.

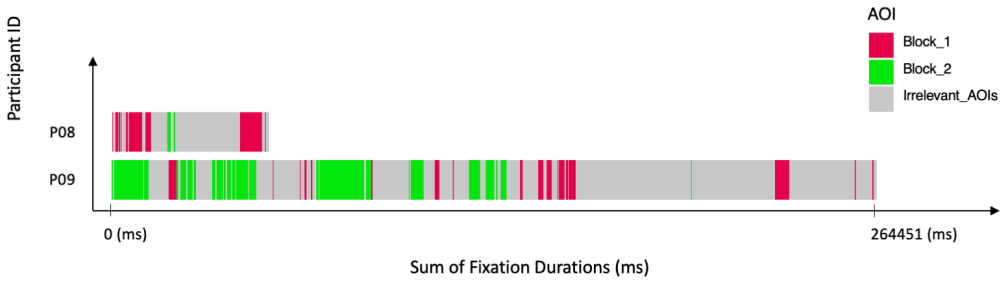


(a) Scarfplot of C-2-A-archimedes.c

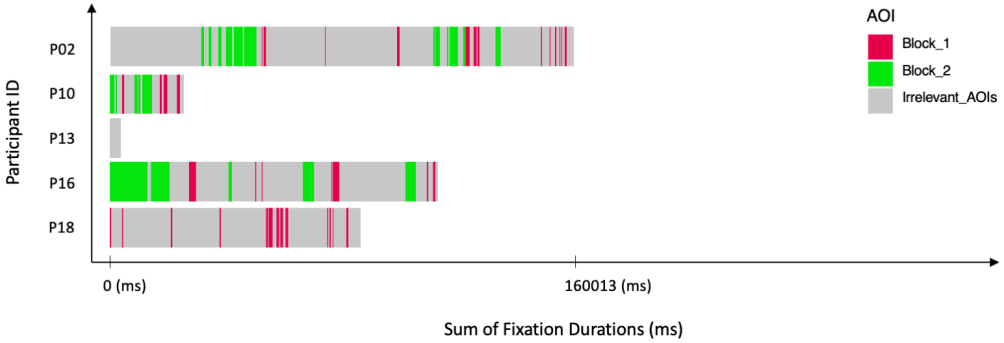


(b) Scarfplot of C-2-B-archimedes.c

Figure 4.15: Scarfplots of Task C-2, File archimedes.c

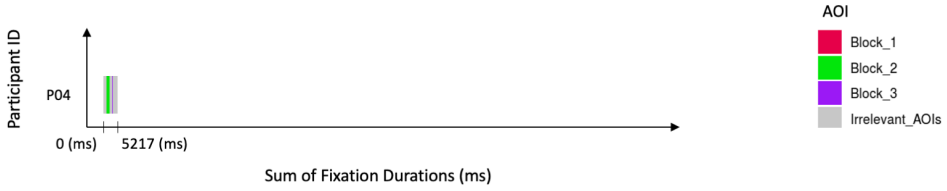


(a) Scarfplot of C-2-A-readinputfile.h

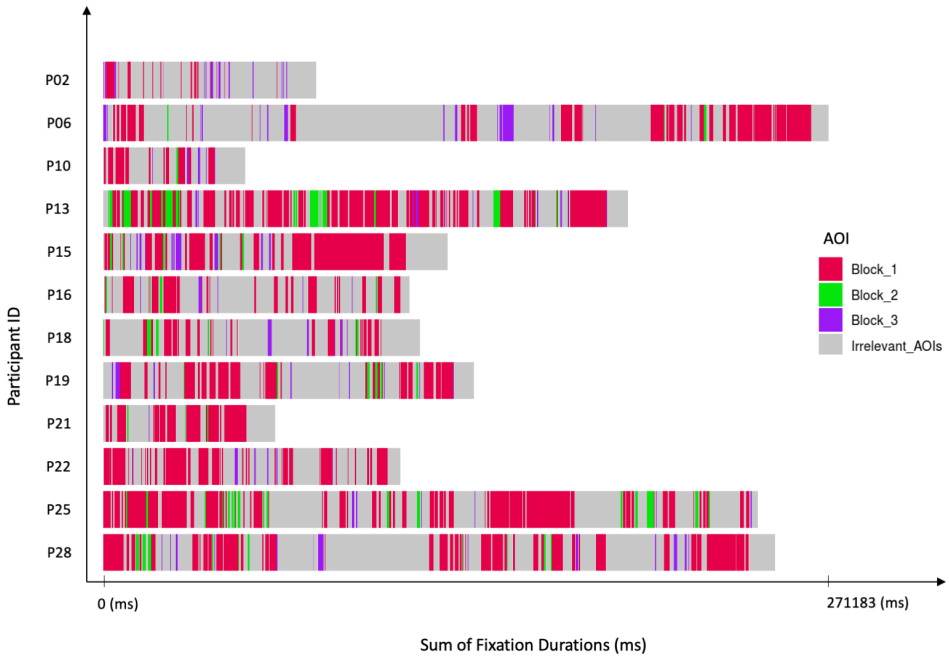


(b) Scarfplot of C-2-B-readinputfile.h

Figure 4.16: Scarfplots of Task C-2, File readinputfile.h



(a) Scarfplot of C-2-A-updating.h

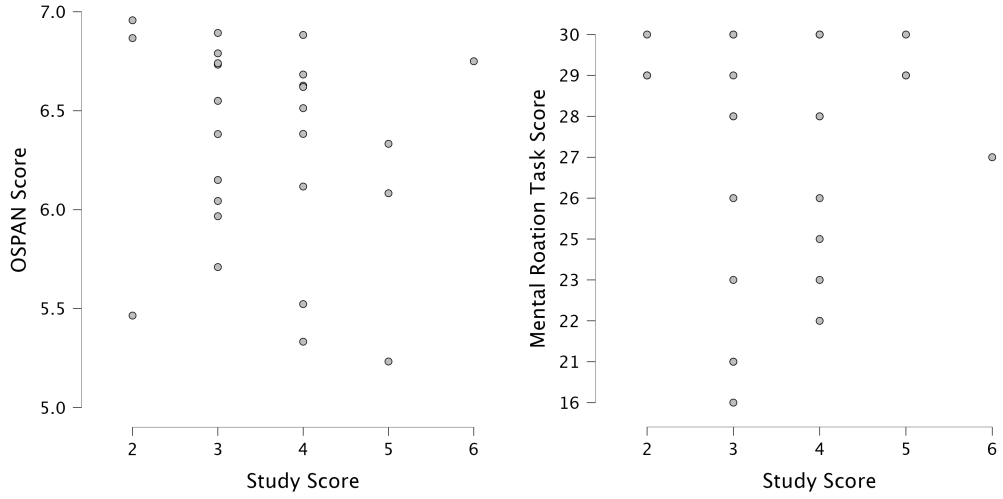


(b) Scarfplot of C-2-B-updating.h

Figure 4.17: Scarfplots of Task C-2, File updating.h

To compare the accuracy of the answers between the non-treatment and merging treatment, we ran the Mann-Whitney U test on the scores of participants in two groups. The test did not show any significant differences ($p = 0.170$, ES = -0.242), indicating that the merging of alarms did not affect the accuracy of the tasks.

Our findings from the two C group tasks give us evidence to test the $MEHV$ and $MEHA$ hypotheses. The evidence from comparing the visual effort on blocks shows us that there are significant differences between the visual effort on blocks for different treatments in task C-1, but we could not find clear and significant differences between metrics in task C-2. Therefore, we cannot reject the null hypothesis, $MEHV_0$, because our findings were inconclusive. Additionally, for hypothesis $MEHA$, the evidence did not show any effects from repositioning on the accuracy of both tasks, so we cannot reject the null hypothesis $MEHA_0$.



(a) Operation Span Task and Study Score Scatterplot

(b) Mental Rotation Task and Study Score Scatterplot

Figure 4.18: Scatterplots of the relationship between the scores

4.5.6 RQ4: Relationship Between Cognitive Tasks and Comprehension Tasks

To answer RQ, we calculated the overall score of the 26 participants from completing all six tasks, and we also calculated the scores for the mental rotation task and operation span task. One participant did not complete the cognitive tasks. Chapter 3, section 3.4.2.1 detail the score calculation method for each cognitive task. The mental rotation task score is calculated out of 30, and the operation span task score is calculated out of 10. To be able to understand the relationship between the study score and cognitive score, we looked at the scatter plots showing the relationship between the mentioned scores. As evident in Figure 4.18, there is not a linear or monotonic relationship between the study score and each of the cognitive tasks, providing no evidence for a relationship between the cognitive and comprehension tasks. Due to this observation, we cannot reject the null hypothesis CAH_0.

4.5.7 Post Questionnaire Results

The results of the post-questionnaire showed us that out of the 27 participants, three rated the study as "Extremely difficult", 22 rated the overall difficulty of the study as "Somewhat difficult", and two rated it as " Neither easy nor difficult". Thirteen participants ranked their programming skills as "Somewhat good", 13 ranked them as "Neither good nor bad", and one participant ranked their programming skill as "Poor". When asked about the difficulties faced during the study, the common answers were difficulties with understanding some C programming functions, fatigue, and difficulties with searching for the right files and areas for the task requiring them to inspect multiple files.

4.6 Threats to Validity

Internal Validity: The participants' proficiency in the C programming language was a threat to the internal validity of the study. In the recruitment step, we asked the participants to state whether they had experience working with the C language. We also asked the participants to complete a proficiency test, but we did not consider the test to exclude some participants, given the already low number of participants. We also made sure that the participants from the online study did not participate in the eye tracking study to avoid any experience effects. The validity of the data from the eye tracker can also be a threat to internal validity, but we mitigated this threat by running calibration several times during the study to ensure that the eye tracker was tracking the gazes correctly and there was no drift.

External Validity: The tasks and their level of difficulty can be a threat to the generalizability of the results. We mitigated this threat by choosing tasks from a combination of open source systems and crafted code which had different difficulty levels and were a good representative of what software developers work on. The low sample size is also a threat to external validity. We had some difficulty finding participants who were willing to work on C programming tasks. Most of our participants were students, and the developers we recruited from industry were not actively working with C and had only worked with it in the past.

Construct Validity: We chose the dependent variables to represent exactly what we were trying to measure to answer our research questions. The study's protocol went through several iterations after which a pilot study was conducted before the main data collection to select number of tasks keeping in mind fatigue effects. Furthermore, to mitigate subject bias, we did not tell the participants about the purpose of the

study, and they did not know about the different repositioning and merging treatments or which group they belonged to.

Conclusion Validity: To ensure conclusion validity, we tested the normality and homogeneity of variance assumptions for all the statistical tests we performed on the data. After checking the assumptions, we used either the Student t-test, Welch's t-test, or the Mann-Whitney test to examine the differences between the two groups.

4.7 Discussion

In response to the research questions stated in Section 4.3, we found that while repositioning and merging static analysis alarms does not have an effect on the accuracy of the tasks, it reduces visual effort in tasks that ask questions about one single file. Our findings about the lack of effect from repositioning and merging on accuracy are in line with our findings from the online study presented in Chapter 3. However, with the analysis of eye tracking data, we observed significant differences between visual effort in treatment and non-treatment tasks when only one file was under inspection. We observed that in tasks B-1, B-2, and C-1, repositioning of the alarm successfully drew the attention of the participants to the more relevant parts of the code for answering the question. Our results showed that the treatments reduced most of the visual effort metrics in the lesser important blocks (Fixation Count, Fixation Duration, and Revisits). We observed that the treatments will cause fewer and shorter fixations on less important blocks, but there is no significant difference in the number and duration of fixations on the important blocks. This indicated that the developers were able to find and learn the important relevant blocks in both treatments, but repositioning or merging the alarms guided them to the important code blocks, and they could find the most relevant information to answer the task

more easily without having to look through many other blocks of the code. The results also showed us that in the tasks that asked the participants to look at only one file, they rarely looked at irrelevant blocks in the code and were able to find the relevant blocks to answer the tasks and focused more on those blocks.

Overall, in the tasks that asked the participant to look at one file to answer the question, while we couldn't find significant differences on the metrics from the entire files, there were changes in visual effort by drawing the attention of the developers to blocks of code that contained information that was pertinent to solving the task. For task C-2, which was the task that required the participants to go through multiple files to answer questions, our findings were different. We observed that the merging treatment did not make any significant difference over the number of files viewed, percentage of relevant files viewed, or the percentage of relevant fixation count or duration over the files. The fixation patterns over important files and blocks for C-2 indicate that without explicit guidance for participants to find relevant files, they were not able to track the variable to their point of variable assignment, and it was difficult for them to find the relevant blocks to answer the question. We believe that more tasks similar to C-2 containing multiple files are needed to draw conclusion about the effects of merging and repositioning in multi-file tasks. Finally, our results did not show a linear relationship between the cognitive tests and the accuracy of the manual inspection of alarms tasks, which can be due to the small set of tasks or the small sample size of participants.

While more empirical research is needed on this topic, based on our study's findings, we can confirm that the repositioning and merging alarms on the static analysis tools can be a useful way to reduce developers' effort while inspecting the alarms. This treatment helped the developers find the more pertinent information to the tasks and helped them focus their attention on the lines of code that were most essential to

inspect and verify the alarm. This can be especially useful in bigger systems and developers who work on bug fixing tasks with the help of static analysis tools every day.

4.8 Conclusion

This eye tracking study investigates the effect of repositioning and merging static analysis alarms on accuracy and visual effort. We also explored the relationship between working memory and spatial cognition with the developers' skills, measured by the accuracy of answering the questions. We recruited 27 participants for this study. The findings indicate that in tasks that only require the participant to look at one file and answer the assertion question, the repositioning and merging of the alarms reduced the visual effort on less relevant blocks. However, when it was required that the participants look through multiple files to solve the task, we did not observe any differences in visual effort. More studies to explore the effects of repositioning and merging on one or multiple files are recommended. We did not observe any effect of the repositioning and merging alarms on the accuracy of the participants' answers. We did not see any relationship between the cognitive scores and the scores in the study. This work provides a methodology and an overview of the experience of conducting an eye tracking study on manually inspecting static analysis alarms in C. The study material is available for other researchers to replicate the study to investigate even further.

Chapter 5

Discussion and Implications

The dissertation presents three studies to examine how developers work with various languages and tools designed to help them detect and fix software defects. The first study on the Alloy specification language explored how novice and non-novice participants performed on bug fixing tasks and looked at their problem solving patterns. The second study was an online survey on the manual inspection of static alarms, with the goal of understanding the effects of repositioning and merging on task performance (accuracy) and problem solving speed. The third study was an eye tracking study exploring the effects of repositioning and merging alarms on visual effort and accuracy in a different set of participants. We examined the spatial cognition abilities and working memory capacities of the participants in all three studies, as well as the relationship between these cognitive skills and task comprehension skills. We discuss the findings and implications in the respective chapters. In this chapter, we discuss the implications of the results of the studies on defect detection and correction for different populations who use or build the languages and tools.

5.1 Alloy Specification Language Study

The empirical study on the differences between novices and non-novices in Alloy showed us that non-novices are significantly more accurate in solving the tasks, and on average, they spend less time working on the models. Furthermore, we found that non-novices performed fewer actions using the analyzer and made fewer edits, and the number of edits and actions positively correlated with accuracy. We also found that participants with higher mental rotation scores performed better on Alloy tasks. We discuss the implications of these results for Alloy novices, teachers, and developers who work on improving Alloy tools.

5.1.1 Implications for Alloy Novices and Instructors

The results showed us that novices work on Alloy models to detect and correct defections in an incremental manner, meaning that they make small changes to the model and visualize the changes using the analyzer to see the effect of their edits. This finding confirms the findings of Li et al. [60]. As the number of edits and actions correlated positively with accuracy, we suggest that beginners follow this practice while working with Alloy models to understand them better. This can help the novices understand the semantics of the models better, get more familiar with the language, and learn how to express the model's semantics using the Alloy language. We suggest that those who teach Alloy in their courses focus more on teaching the semantics of the language and problems as well as the features of the analyzer. This is because novices did not have a problem finding and fixing the syntactic defects due to the analyzer's syntax debugging feature. Since novices rely on executing the model to solve the bug detection and correction tasks, we also suggest that the educators spend significant

time teaching the students the semantics of visualizations of model instances and how they can help the students solve the tasks at hand.

5.1.2 Implications for Alloy Bug Triaging

Bug triaging is the process of prioritizing bug reports/issues for developers working on a project. It is a time consuming process and efficiency in bug triaging can help productivity and move the project forward. In our study, we found that participants with higher mental rotation tasks did significantly better on the Alloy tasks. Based on this finding, we believe that assigning people with higher spatial cognition to fixing highly critical defects in Alloy can increase the chance of the defect getting fixed in a more accurate and timely manner.

5.1.3 Improvements Suggested for the Alloy Analyzer

The Alloy Analyzer provides visualization and the description of instances, and the study showed us that it is used by both novices and non-novices to look for defects and to understand the models. We suggest changes to the Alloy Analyzer, such as adding debugging capabilities to facilitate defect detection. Methods such as automatic bug localization in Alloy proposed by Zheng et al. [137] can be used to detect potential faults in Alloy models, and the results of these tools can help the users in finding and correcting faults. Furthermore, due to novices relying on the analyzer, automated compilation and execution of the model given a change can be helpful to them as they work on an Alloy model.

5.2 Studies on the Manual Inspection of Static Analysis Alarms

We conducted two separate studies on the manual inspection of static analysis alarms to examine the effects of repositioning and merging the alarms on the developers' performance. The first study was done through an online questionnaire hosted on Qualtrics and distributed through Mechanical Turk and by link sharing to social media and the researchers' contacts, and an eye tracking study with 27 participants was conducted to get more insight into the developers' cognitive processes while they worked on these tasks and uncover detailed information that could not be extracted from online submissions. The results of the online questionnaire and the eye tracking study showed us that repositioning and merging did not have any effects on the accuracy of the manual inspection tasks. However, the eye tracking study showed that repositioning and merging alarms reduce the visual effort on the less relevant blocks when the alarms are on one single file. We did not observe a significant difference in the most important blocks, indicating that the participants eventually found the relevant information but after more visual effort in non-treatment tasks. Furthermore, we did not observe the same pattern in the tasks that required the participants to traverse multiple files to answer the question. We discuss the implications of these studies.

5.2.1 Implications for Users of Static Analysis Tools

Using repositioning and merging of static alarms can address many of the issues stated by developers in interviews done by Johnson et al. [15], such as the high number of alarms and the presentation of the alarms. Merging of the alarms reduces the number of them presented to the developers, and the repositioning of the alarms presents

the alarms in more relevant places on the code, closest to the statements that are the generation points of the alarms. Based on the results of the eye tracking study, repositioning and merging the alarms is useful to decrease the developers' efforts when they are working with static analysis tools. The repositioning and merging treatments helped the developers focus on the most relevant parts of the code to solve the tasks. This finding confirms the effectiveness of the repositioning and merging treatment in reducing effort in developers. This also gives added evidence to support users via better future tools and methods in further tasks related to static analysis tools.

5.2.2 Implications for Developers of Static Analysis Tools

We encourage the developers and teams that build static analysis tools to consider the empirical evidence about the usability of the alarms and apply the methods proposed to improve the tools for users. If the developers do not have the necessary user friendly tools to find defects and fix them, it increases the manual effort of defect detection and the chances of burn out [138]. Static analysis tools can be useful for developers with different experience levels, and improving their usability helps developers write higher quality code and build software with fewer defects.

Chapter 6

Conclusions and Future Work

The studies presented in this dissertation examined developers' behavior and cognitive processes while working on defect detection and correction tasks. Overall, we recruited 306 developers to participate in all the studies presented.

To the best of our knowledge, we conducted the first empirical study on defect detection in Alloy to compare novices and non-novices, and found clear differences between novice and non-novice work patterns. Based on these differences, we suggest some improvements to the Alloy Analyzer to better support novices. We also conducted the first eye tracking study on C code with the goal of examining the effects of repositioning and merging static analysis alarms on the manual inspection of the alarms. An online questionnaire-based study was also conducted prior to the eye tracking study for static analysis manual alarm verification tasks. The goal of conducting the eye tracking study for static analysis alarm verification was to help explain the results from the online study. This shows that using multiple methods of data collection (i.e., in our case eye tracking and online questionnaires) is preferred when seeking to learn more about observed behavior. The eye tracking study found a significant reduction in visual effort (not detected in the online study) on the less important blocks of code after repositioning and merging treatment on one single file. The complete replication packages for verifiability of these studies are available online.

With regards to future work, for the study on the Alloy specification language, an extension of the study with eye tracking, monitoring the participants, and extending the logging capability of the analyzer to keep track of the time between each edit can give us more understanding of the participants' problem solving process while working on the Alloy tasks. The replication package can help other researchers conduct the same or similar studies. For studying the effects of different methods to improve the usability of static analysis alarms, more studies using various types of tasks in different programming languages can be very useful, especially with introducing different tasks that require the participants to look at multiple files to inspect the alarms. In addition, due to the amount of spam generated from online studies such as Mechanical Turk, future studies should plan on incorporating attention check tasks (a random question that requires a human to solve a simple task) within the questionnaire to improve detecting spam entries. Overall, given the results of the studies conducted, we believe more work is warranted in this area.

Appendix A

Alloy Specification Language Study Task Files and Supplemental Material

The appendix presents the related documents and the study tasks. The replication package for this study is available at:

`https://osf.io/5p6e4/?view_only=861a8de10ce249fa92eda4b9ac7776cd`

A.1 Documents

The documents section includes the following:

1. IRB Approval Form
2. Alloy study informed consent document
3. Pre-questionnaire
4. Post-questionnaire
5. ReadMe file for the participants

University of Nebraska Lincoln

Official Approval Letter for IRB project #19639 - New Project Form

February 13, 2020

[redacted]

[redacted]

IRB Number: 20200219639EP
Project ID: 19639
Project Title: An Exploratory Study Assessing the Alloy Specification Language

Dear Bonita:

This letter is to officially notify you of the approval of your project by the Institutional Review Board (IRB) for the Protection of Human Subjects. It is the Board's opinion that you have provided adequate safeguards for the rights and welfare of the participants in this study based on the information provided. Your proposal is in compliance with this institution's Federal Wide Assurance 00002258 and the DHHS Regulations for the Protection of Human Subjects under the 2018 Requirements at 45 CFR 46.

o Review conducted using expedited review categories 6 & 7 at 45 CFR 46.110
o Date of Approval: 02/13/2020
o Date of Expedited review: 11/26/2020
o Date of Acceptance of Revisions: 02/13/2020
o Funding (Grant congruency, OSP Project/Form ID and Funding Sponsor Award Number, if applicable): NSF; OSP Project ID 48500, Form ID 131543; Grant Congruency Review 2/13/2020
o Consent waiver : N/A
o Review of specific regulatory criteria (contingent on funding source): 45 CFR 46
o Subpart B, C or D review : N/A

You are authorized to implement this study as of the Date of Final Approval: 02/13/2020. This approval is Valid Until: 02/12/2021.

We wish to remind you that the principal investigator is responsible for reporting to this Board any of the following events within 48 hours of the event:
* Any serious event (including on-site and off-site adverse events, injuries, side effects, deaths, or other problems) which in the opinion of the local investigator was unanticipated, involved risk to subjects or others, and was possibly related to the research procedures;
* Any serious accidental or unintentional change to the IRB-approved protocol that involves risk or has the potential to recur;
* Any protocol violation or protocol deviation
* An incarceration of a research participant in a protocol that was not approved to include prisoners
* Any knowledge of adverse audits or enforcement actions required by Sponsors
* Any publication in the literature, safety monitoring report, interim result or other finding that indicates an unexpected change to the risk/benefit ratio of the research;
* Any breach in confidentiality or compromise in data privacy related to the subject or others; or
* Any complaint of a subject that indicates an unanticipated risk or that cannot be resolved by the research staff.

Any changes to the project, including reduction of procedures, must be submitted and approved prior to implementation. A change request form must be submitted to initiate the review of a modification.

For projects which continue beyond one year from the starting date, an annual update of the project will be required by informing the IRB of the status of the study. The investigator must also advise the Board when this study is finished or discontinued by completing the Final Report form via NUgrant.

If you have any questions, please contact the IRB office at 402-472-6965.

Sincerely,

[redacted]

Becky R. Freeman, CIP
for the IRB

Full Accreditation

University of Nebraska Lincoln

IRB #:

Formal Study Title:
An Exploratory Study Assessing the Alloy Specification Language

Authorized Study Personnel
Principle Investigator: Dr. Bonita Sharif Office: 402-472-5084
Secondary Investigator: Ms. Niloofar Mansoor, Hamid Bagheri

Key Information:

If you agree to participate in this study, the project will involve:

- Adults between 19 years of age and older
- Procedures will include completing programming tasks
- 1 visit is required
- These visits will take 2-3 hours total
- There are minimal risks associated with this study such as that of mental distress that could occur during programming. Tasks in this study are timed and you may feel uncomfortable with timed tasks, but it is the easiest method to electronically track time in a fair and consistent way.
- You will be paid [redacted] for your participation. The payment will be in the form of a gift card.
- You will be provided a copy of this consent form.

Invitation
You are invited to take part in this research study. The information in this form is meant to help you decide whether or not to participate. If you have any questions, please ask.

Why are you being asked to be in this research study?
You are being asked to be in this study because you are either a student who knows how to program in a computer language, an industry professional, or a faculty member whose expertise and research relates to Computer Science. You must be 19 years of age or older in order to participate in this study.

What is the reason for doing this research study?
The purpose of this study is to understand how developers comprehend a declarative specification language. This research is designed to better understand these languages.

What will be done during this research study?
You will be asked to sit in front of a computer monitor and do some programming tasks. We will collect eye movement data via an eye tracker that sits under the monitor. The study will take 2 hours to complete. You will be given a time schedule of when to appear in the lab in Schorr Center. A recording of the screen during the experiment will be kept, but we will not record any videos or audio of participants. Once before and once after performing the tasks, you will be asked to take surveys about the study and the language.

How will my data be used?
The recordings will be used to examine how one goes about solving the task i.e., using the verbal responses and using the screen recording. They will not be identifying i.e.; we will not use specific videos in any form for publication. We will store them on a password protected drive. No video of the person will be recorded. Only the screen will be recorded. Any personal information that could identify you will be removed before the data are shared. In addition, de-identified data will be made available online as a dataset artifact.
This study will involve the collection of private information (name, dates, etc.). Your information could be used or distributed to another researcher for future research studies without an additional informed consent from you. Identifiers (name, dates, etc.) will be removed prior to being distributed.

What are the possible risks of being in this research study?
There are minimal risks associated with this study such as that of mental distress that could occur during programming. Tasks in this study are timed and you may feel uncomfortable with timed tasks, but it is the easiest method to electronically track time in a fair and consistent way.

What are the possible benefits to you?
You may learn about the Alloy language by participating in this study. Other than possibly contributing to scientific knowledge in the field, there are otherwise no benefits for participation. However, you may not get any benefit from being in this research study.

What are the possible benefits to other people?
The benefits to science and/or society may include better understanding of how to help design new programming languages and understand/improve current ones.

What will being in this research study cost you?
There is no cost to you to be in this research study.

Will you be compensated for being in this research study?
You will receive [redacted] for completing this study. The payment will be in the form of a gift card.

What should you do if you have a problem during this research study?

Your welfare is the major concern of every member of the research team. If you have a problem as a direct result of being in this study, you should immediately contact one of the people listed at the beginning of this consent form.

How will information about you be protected?
Reasonable steps will be taken to protect your privacy and the confidentiality of your study data. The data will be stored electronically through a secure server and will only be seen by the research team during the study and will be archived after the study is complete. The only persons who will have access to your research records are the study personnel, the Institutional Review Board (IRB), and any other person, agency, or sponsor as required by law. The information from this study may be published in scientific journals or presented at scientific meetings but the data will be reported as group or summarized data and your identity will be kept strictly confidential.

What are your rights as a research subject?
You may ask any questions concerning this research and have those questions answered before agreeing to participate in or during the study.

For study related questions, please contact the investigator(s) listed at the beginning of this form.

For questions concerning your rights or complaints about the research contact the Institutional Review Board (IRB):

- Phone: 1(402)472-6965
- Email: irb@unl.edu

What will happen if you decide not to be in this research study or decide to stop participating once you start?
You can decide not to be in this research study, or you can stop being in this research study ("withdraw') at any time before, during, or after the research begins for any reason. Deciding not to be in this research study or deciding to withdraw will not affect your relationship with the investigator or with the University of Nebraska-Lincoln. You will not lose any benefits to which you are entitled.

Documentation of informed consent
You are voluntarily making a decision whether or not to be in this research study. Signing this form means that (1) you have read and understood this consent form, (2) you have had the consent form explained to you, (3) you have had your questions answered and (4) you have decided to be in the research study. You will be given a copy of this consent form to keep.

Participant Feedback Survey
The University of Nebraska-Lincoln wants to know about your research experience. This 14 question, multiple-choice survey is anonymous. This survey should be completed after your participation in this research. Please complete this optional online survey at:

http://bit.ly/UNLresearchfeedback.

Participant Name:

_____
(Name of Participant: Please print)

Participant Signature:

_____

Signature of Research Participant

_____

Date

Pre Questionnaire

This form presents questions of the pre questionnaire for the Alloy user study. You have chosen to participate in this study, and your answers to these questions help us better understand you and your skills.

* Required

1. What is your participant ID? *

2. What is your age group? *

Mark only one oval.

- ◯ 18-20
- ◯ 21-25
- ◯ 26-30
- ◯ 31-35
- ◯ 36-40
- ◯ 40+

3. What is your gender? *

Mark only one oval.

- ◯ Female
- ◯ Male
- ◯ Non-binary
- ◯ Other: _____

4. At what institution are you completing this study? *

5. What degrees do you have or are currently pursuing? *

Check all that apply.

- [ ] Bachelor's Degree
- [ ] Master's Degree
- [ ] Doctorate

6. What is (was) your major? *

_____

7. If you are a student, on a 4.0 scale, approximately what is your academic GPA? *

_____

8. What is your country of origin? *

_____

9. What is your primary language? *

_____

10. If English is not your native language, on a scale from 0 to 5, how fluent would you consider yourself to be in English?

Mark only one oval.

Not Fluent

0

1

2

3

4

5

Completely Fluent

Post Questionnaire

This questionnaire helps us know more about your experience participating in the study, and have more information about your background and your skills.

* Required

1. What is your participant ID? *

2. Did you have any trouble completing the tasks? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No

3. If you answered yes on the previous question, please explain what issues you had with the tasks.

4. Did you have sufficient time to complete the tasks? *

   Mark only one oval.

   - ◯ Yes
   - ◯ No

5. Rate your programming skills. *

Mark only one oval.

Poor

1

2

3

4

5

Excellent

6. Rate your software design skills. *

Mark only one oval.

Poor

1

2

3

4

5

Excellent

7. Rate your knowledge in set theory. *

Mark only one oval.

Poor

1

2

3

4

5

Excellent

8. Rate your knowledge in first order logic. *

Mark only one oval.

Poor

1

2

3

4

5

Excellent

9. Rate your skill in object-oriented languages. *

Mark only one oval.

Poor

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Excellent

10. Rate your level of comfort in using and comprehending the Alloy syntax. *

Mark only one oval.

Not comfortable

1 ◯

2 ◯

3 ◯

4 ◯

5 ◯

Very comfortable

11. Rate your level of comfort in using the Alloy Analyzer. *

Mark only one oval.

	Not comfortable					Very comfortable
	1	2	3	4	5	

12. How much experience do you have with programming languages? *

Mark only one oval.

- Less than 1 year
- Between 1 and 3 years
- Between 3 and 5 years
- More than 5 years

13. How much experience do you have with specification languages? *

Mark only one oval.

- Less than 1 year
- Between 1 and 3 years
- Between 3 and 5 years
- More than 5 years

14. How many years of experience do you have in working with Alloy? *

Mark only one oval.

- ◯ Less than 1 year
- ◯ Between 1 and 3 years
- ◯ Between 3 and 5 years
- ◯ More than 5 years

15. Which one of the following languages/tools have you worked with before? *

Check all that apply.

- ☐ Alloy
- ☐ Z
- ☐ SPIN
- ☐ Java PathFinder
- ☐ NuSMV
- ☐ Z3
- ☐ TLA+
- ☐ None
- ☐ Other: _____

16. Do you have any other comments or suggestions about the study?

_____

_____

_____

_____

_____

A guide to the experiment

An Overview.

Thank you for volunteering to participate in our study of the Alloy specification language. The goal of this experiment is to see how you find and fix issues in the Alloy specifications that are provided to you given their natural language specification, and how you will write your own Alloy specification for a small problem. You are also asked to complete a pre and post questionnaire, two short cognitive tasks, and record your start and finish time for the Alloys tasks.

Please make sure to do each task in one sitting, as the start and finish time are important factors.

Before beginning the study, make sure that you have the Java Development Kit and Python 2.7 installed on your computer. You will need java for running the Alloy Analyzer and Python 2.7 for running one of the cognitive tasks. You can download the JDK at this link: https://www.oracle.com/java/technologies/javase-downloads.html
You can download Python 2.7 at this link:
https://www.python.org/downloads/release/python-2718/

The following are the steps to completing the tasks for this study.

The Process.

Step 1: Fill out the pre-questionnaire: https://forms.gle/iwxxzt6EuznGTLMi9

Step 2: Go to the **Cognitive Tasks** folder.

Step 3: Open the **Task1** folder and open the file *MRT.jar*. Follow the instructions for this cognitive task that appear on the screen. Complete this task only once. There will be a generated results file in the folder, please keep that file in the folder.

Step 4 (Windows): Open Command line by clicking on Start, and typing Command Prompt in the search bar. Click on the Command Prompt program when it appears. Navigate to the *Task2* folder using the *cd* command. Make sure that you have Python 2.7 installed. (You can check this by typing python --version in the command line. It should print Python 2.7.18)
Use this command to run the program:

- python pyspantask.py

Step 4 (Mac or Linux): Open the terminal and navigate to the *Task2* folder using the *cd* command. Make sure that you have Python 2.7 installed. Use this command to run the program:

- python pyspantask.py

Step 5: Follow the instructions for this cognitive task that appear on the screen. Complete the task only once. There will be a generated results file in the folder, please keep that file in the folder.

Step 6: Open the Alloy Analyzer from the folder named Alloy. The file is named: *org.alloytools.alloy.dis.logging.capability.jar*

Step 7: You can open the Alloy specification files (files with the *.als* extension) using the Open icon in the Alloy Analyzer.

Step 8:

1. Open the file located in the **Sample Task** folder, named **filesystem.als**.

2. Once you have your Alloy specification file open, you can open the **spec.txt** file from the **Sample Task** folder, which includes the natural language specification of the problem or system described in the Alloy file.

3. Read through the specification file and study the Alloy model. The bugs that are introduced in this sample task are revealed in the file, using comments starting with //BUG. You can expect similar types of bugs in the real tasks.

Step 9:

Once you are done studying the sample task, you are going to work on the tasks included in the *Tasks* folder.

Go to the *Bug Finding Tasks* subfolder of the *Tasks* folder. There are folders that are named *Model 1, Model 2, Model 3*. Each folder includes an Alloy file, and a natural language specification text file.

Note: While the natural language specification provided to you is completely correct, do not make any assumptions about the correctness of the Alloy specification. The bugs can be anywhere in the model.

Note: You are free to modify the specification to fix the problems you see in the model. There are between 3-4 bugs in each model. Modify the specification with the correct code and save the Alloy file.

Step 10:

1. In the *Bug Finding Tasks* subfolder of the *Tasks* folder, go to the folder *Model 1*.

2. Open *specification.txt* text file in the folder *Model 1*. This file provides the natural language specification of the model, and some questions that we want you to answer.

3. Open the model *grade_faulty.als* in the Alloy Analyzer.

4. Record your start time and write it in the *specification.txt* document.

5. Study the natural language specification and the Alloy specification. You can use all the features of the Alloy Analyzer to see the different generated models.

6. Find the problems in the Alloy specification and change the model in a way that it would completely correspond to the natural language specification.

7. When you are finished with finding and fixing the problems, save your Alloy model to include the changes that you made, and record your finish time in the *specification.txt* file.

8. Record the confidence in your answer and state your comments on your answer in the *specification.txt* file.

9. Make sure to save the modified versions of both the *specification.txt* and *grade_faulty.als*.

Step 11:

1. In the *Bug Finding Tasks* subfolder of the *Tasks* folder, go to the folder *Model 2*.

2. Open *specification.txt* text file in the folder *Model 2*. This file provides the natural language specification of the model, and some questions that we want you to answer.

3. Open the model *BalancedBST_faulty.als* in the Alloy Analyzer.

4. Record your start time and write it in the *specification.txt* document.

5. Study the natural language specification and the Alloy specification. You can use all the features of the Alloy Analyzer to see the different generated models.

6. Find the problems in the Alloy specification and change the model in a way that it would completely correspond to the natural language specification.

7. When you are finished with finding and fixing the problems, save your Alloy model to include the changes that you made, and record your finish time in the *specification.txt* file.

8. Record the confidence in your answer and state your comments on your answer in the *specification.txt* file.

9. Make sure to save the modified versions of both the *specification.txt* and *BalancedBST_faulty.als*.

Step 12:

1. In the *Bug Finding Tasks* subfolder of the *Tasks* folder, go to the folder *Model 3*.

2. Open *specification.txt* text file in the folder *Model 3*. This file provides the natural language specification of the model, and some questions that we want you to answer.

3. Open the model *farmer_faulty.als* in the Alloy Analyzer.

4. Record your start time and write it in the *specification.txt* document.

5. Study the natural language specification and the Alloy specification. You can use all the features of the Alloy Analyzer to see the different generated models.

6. Find the problems in the Alloy specification and change the model in a way that it would completely correspond to the natural language specification.

7. When you are finished with finding and fixing the problems, save your Alloy model to include the changes that you made, and record your finish time in the *specification.txt* file.

8. Record the confidence in your answer and state your comments on your answer in the *specification.txt* file.

9. Make sure to save the modified versions of both the *specification.txt* and *farmer_faulty.als*.

Step 13:

1. After you are done fixing the Alloy specification bugs, go back into the *Tasks* folder and go into the *Complete Your Own Model* folder.

2. Open the file *LinkedList.als* using the Alloy Analyzer. This is a partially completed specification of a Linked List structure.

3. Open *specification.txt* text file in the folder *Complete Your Own Model*. This file provides the natural language specification of the model, and some questions that we want you to answer.

4. Record your start time and write it in the *specification.txt* document.

5. Study the natural language specification and the Alloy specification. You can use all the features of the Alloy Analyzer to see the different generated models.

6. Complete the Alloy specification in a way that it would completely correspond to the natural language specification.

7. When you are finished with writing your model, save your Alloy model to include the changes that you made, and record your finish time in the *specification.txt* file.

8. Record the confidence in your answer and state your comments on your answer in the *specification.txt* file.

9. Make sure to save the modified versions of both the *specification.txt* and *LinkedList.als.*

Step 14:

Fill out the post questionnaire: https://forms.gle/NyyPREYEuBLBXsvs5

Zip up the folder and email it to niloofar@huskers.unl.edu, preferably with the title: Alloy Empirical Study.

A.2 Model 1: grade.als

The following pages contain the natural language specification and Alloy model for *grade.als*.

Specification for Student Grading – grade_faulty.als

a) Record start time here :

b) Change the model such that the specification corresponds to the following natural language specification:

This Alloy specification describes a gradebook and an important policy about grading. The gradebook has records of students and professors.

- Classes have one faculty instructor (a professor) and can have a number of students working as teaching assistants (TAs).
- Each assignment that is graded in this gradebook is associated with only one class, and it is assigned to the students enrolled in that class.
- The grading policy is enforced to ensure that no student can grade their own assignment. Only the professor and the TAs of the class are allowed to grade the assignments.
- We want to check and see if this important policy holds, or we can find any instances that break this policy. Ideally, we don't want to see any instances breaking the policy.

c) Record end time here :

d) Save your model

e) Record the confidence in your answer: Not confident Somewhat Confident Very Confident

f) Please state your comments on your answer:

```
1  abstract sig Person {}
2
3  sig Student extends Person {}
4
5  sig Professor extends Person {}
6
7  sig Class {
8    assistant_for: set Student,
9    instructor_of: one Professor
10 }
11
12 sig Assignment {
13   associated_with: set Class,
14   assigned_to: some Student
15 }
16
17 pred PolicyAllowsGrading(s: Person, a: Assignment) {
18   s in a.associated_with.assistant_for || s in a.associated_with.instructor_of
19   s in a.assigned_to
20 }
21
22 assert NoOneCanGradeTheirOwnAssignment {
23   all s : Person | all a: Assignment | PolicyAllowsGrading[s]
24   implies not s in a.assigned_to
25 }
26
27 check NoOneCanGradeTheirOwnAssignment
```

Listing A.1: grade.als Model

A.3 Model 2: balancedBST.als

The following pages contain the natural language specification and Alloy model for *balancedBST.als*.

Specification for Balanced Binary Search Tree – balancedBST.als

a) Record start time here :

b) Change the model such that the specification corresponds to the following natural language specification:

This Alloy specification describes a balanced binary search tree. A binary search tree is a data structure that keeps its elements in a specific order.

- For each parent node, nodes with values smaller than the parent node's value are to the left, and nodes with values greater than the parent node's value are to the right.
- An empty subtree is automatically balanced.
- A non-empty BST is balanced if both the left and right subtrees are balanced, and the difference between heights of the right and left subtree is not more than 1.
- The model should be able to generate only correct balanced binary search trees.

c) Record end time here :

d) Save your model

e) Record the confidence in your answer: Not confident Somewhat Confident Very Confident

f) Please state your comments on your answer:

```
one sig BinaryTree {
  root: lone Node
}
sig Node {
  left, right: lone Node,
  elem: Int
}
// All nodes are in the tree.
fact Reachable {
  Node = BinaryTree.root*.(left + right)
}
// Part (a)
fact Acyclic {
  all n : Node {
    // There are no directed cycles, i.e., a node is not reachable
    // from itself along one or more traversals of left or right.
    n !in n.^(left + right)

    // A node cannot have more than one parent.
    some n.~(left + right)

    // A node cannot have another node as both its left child and
    // right child.
    no n.left & n.right
  }
}
// Part (b)
pred Sorted() {
  all n: Node {
    // All elements in the n's left subtree are smaller than the n's elem.
    some n.left =>n.left.elem<n.elem

    // All elements in the n's right subtree are bigger than the n's elem.
    some n.right =>n.right.elem>n.elem
  }
}
// Part (c.1)
pred HasAtMostOneChild(n: Node) {
  // Node n has at most one child.
  no n.left || no n.right
}
// Part (c.2)
fun Depth(n: Node): one Int {
  // The number of nodes from the tree's root to n.
  #{n.*~(left + right)}
}
// Part (c.3)
pred Balanced() \{
  all n1, n2: Node {
    // If n1 has at most one child and n2 has at most one child,
    // then the depths of n1 and n2 differ by at most 1.
    // Hint: Be careful about the operator priority.
    (HasAtMostOneChild[n1] && HasAtMostOneChild[n2]) =>
    (let diff = minus[Depth[n1], Depth[n2]] | -1 <= diff || diff <= 1)
  }
}
pred RepOk() {
  Sorted
  Balanced
}
run RepOk for 5
```

Listing A.2: balancedBST.als Model

A.4 Model 3: farmer.als

The following pages contain the natural language specification and Alloy model *farmer.als*.

Specification for River Crossing Puzzle – farmer.als

a) Record start time here :

b) Change the model such that the specification corresponds to the following natural language specification:

This Alloy specification describes the classic river crossing puzzle. A farmer is carrying a fox, a chicken, and a sack of grain.

- He must cross a river using a boat that can only hold the farmer and at most one other thing. If the farmer leaves the fox alone with the chicken, the fox will eat the chicken; and if he leaves the chicken alone with the grain, the chicken will eat the grain.

- How can the farmer bring everything to the far side of the river intact? When you run the Alloy specification, you should see a valid instance of the solved puzzle.

c) Record end time here :

d) Save your model

e) Record the confidence in your answer: Not confident Somewhat Confident Very Confident

f) Please state your comments on your answer:

```
module farmer
open util/ordering[State] as ord
/*
* The farmer and all his possessions will be represented as Objects.
* Some objects eat other objects when the Farmer's not around.
*/
abstract sig Object { eats: set Object }
one sig Farmer, Fox, Chicken, Grain extends Object {}
/*
* Define what eats what when the Farmer' not around.
* Fox eats the chicken and the chicken eats the grain.
*/
fact eating { eats = Fox->Chicken + Chicken->Grain }
/*
* The near and far relations contain the objects held on each
* side of the river in a given state, respectively.
*/
sig State {
   near: set Object,
   far: set Object
}
/*
 * In the initial state, all objects are on the near side.
 */
fact initialState {
   let s0 = ord\first |
     s0.near = Object & no s0.far
}
/*
 * Constrains at most one item to move from 'from' to 'to'.
 * Also constrains which objects get eaten.
 */
pred crossRiver [from, from', to, to': set Object] {
  // either the Farmer takes no items
  ( from' = from - Farmer - from'.eats &&
    to' = to + Farmer ) ||
  // or the Farmer takes one item
 (one item: from - Farmer | {
    from' = from - Farmer - item
    to' = to - to.eats + Farmer + item
  })
}
/*
 * crossRiver transitions between states
 */
fact stateTransition {
  all s: State, s': ord/next[s] {
    Farmer in s.near =>
      crossRiver[s.near, s'.near, s.far, s'.far] else
      crossRiver[s'.far, s.far, s'.near, s.near]
  }
}
/*
 * the farmer moves everything to the far side of the river.
 */
pred solvePuzzle {
     ord/last.far = Object
}

run solvePuzzle for 8 State expect 1
```

Listing A.3: farmer.als Model

Appendix B

Static Alarms Study Task Files and Supplemental Material

The appendix presents the related documents and the study tasks. The replication package and supplemental material for the online survey are available at:

`https://osf.io/u32hj/?view_only=fe862992cef34476a2407bda6ec8b731`

The replication package and supplemental material for the eye tracking study are available at:

`https://osf.io/92b7w/?view_only=5d1e162860b24d5badad350b2da2f598`

B.1 Documents

The documents section includes the following:

1. IRB Approval Form
2. Informed consent document
3. Qualtrics survey including the proficiency questions and pre and post questionnaires

University of Nebraska Lincoln

Official Approval Letter for IRB project #19657 - New Project Form

October 19, 2020

[redacted]

[redacted]

IRB Number: 20201019657EX
Project ID: 19657
Project Title: Assessing Static Alarm Warning Messages

Dear Bonita:

This letter is to officially notify you of the certification of exemption of your project for the Protection of Human Subjects. Your proposal is in compliance with this institution's Federal Wide Assurance 00002258 and the DHHS Regulations for the Protection of Human Subjects at 45 CFR 46 2018 Requirements and has been classified as exempt. Exempt categories are listed within HRPP Policy #4.001: Exempt Research available at: https://research.unl.edu/researchcompliance/policies-procedures/.

o Date of Final Exemption: 10/19/2020
o Certification of Exemption Valid-Until: 10/19/2025
o Review conducted using exempt category 3(ii) at 45 CFR 46.104
o Funding: Federal, National Science Foundation, Grant Congruency performed by RW on 10/19/2020, OSP Project 48500, Form ID 126582.

This exemption determination is for UNL involved faculty, staff, and students only. Per UNL HRPP Policy #3.012: IRB Approval of Multi-site or Cooperative Research, when a project is determined to be Exempt Research, the UNL IRB will not act as the Reviewing IRB or cede review of a project to another institution. As such, your collaborators will need to work with their own institutions IRB office to make their own determinations based on their policies.
Please allow sufficient time for the official IRB approval letter to be available within NUgrant.

We wish to remind you that the principal investigator is responsible for reporting to this Board any of the following events within 48 hours of the event:
* Any serious event (including on-site and off-site adverse events, injuries, side effects, deaths, or other problems) which in the opinion of the local investigator was unanticipated, involved risk to subjects or others, and was possibly related to the research procedures;
* Any serious accidental or unintentional change to the IRB-approved protocol that involves risk or has the potential to recur;
* Any protocol violation or protocol deviation
* An incarceration of a research participant in a protocol that was not approved to include prisoners
* Any knowledge of adverse audits or enforcement actions required by Sponsors
* Any publication in the literature, safety monitoring report, interim result or other finding that indicates an unexpected change to the risk/benefit ratio of the research;
* Any breach in confidentiality or compromise in data privacy related to the subject or others; or
* Any complaint of a subject that indicates an unanticipated risk or that cannot be resolved by the research staff.

This project should be conducted in full accordance with all applicable sections of the IRB Guidelines and you should notify the IRB immediately of any proposed changes that may affect the exempt status of your research project. You should report any unanticipated problems involving risks to the participants or others to the Board.

If you have any questions, please contact the IRB office at 402-472-6965.

Sincerely,

[redacted]

Rachel Wenzl, CIP
for the IRB

Full Accreditation

University of Nebraska Lincoln

IRB #:

Formal Study Title:
Assessing Static Alarm Warning Messages via Eye Tracking

Authorized Study Personnel
Principle Investigator: Dr. Bonita Sharif Office: 402-472-5084
Secondary Investigator: Ms. Niloofar Mansoor Cell: [redacted]

Key Information:

If you agree to participate in this study, here's some information you need to know:

- Adults between 19 years of age and older
- Procedures will include completing questionnaires on-line and completing programming tasks
- 1 visit is required
- These visits will take 1.5-2 hours total
- There are minimal risks associated with this study such as that of mental distress that could occur during programming. Tasks in this study are timed and you may feel uncomfortable with timed tasks, but it is the easiest method to electronically track time in a fair and consistent way.
- You will be paid [redacted] for your participation. The payment will be in the form of a gift card.
- You will be provided a copy of this consent form

Invitation
You are invited to take part in this research study. The information in this form is meant to help you decide whether or not to participate. If you have any questions, please ask.

Why are you being asked to be in this research study?
You are being asked to be in this study because you are either a student who knows how to program in a computer language or a professional developer working in industry. You must be 19 years of age or older in order to participate in this study.

What is the reason for doing this research study?
The purpose of this study is to understand how developers comprehend programming languages. This research is designed to better understand these languages.

What will be done during this research study?
You will be asked to sit in front of a computer monitor and do some programming tasks. We will collect eye movement data via an eye tracker that sits under the monitor. The study will take 2 hours to complete. You will be given a time schedule of when to appear for the study. Audio and video will be recorded during the experiment. At the end of the tasks, you will be asked to take a survey about programming languages. IP addresses are stored to provide continuity to responses and merge data with eye tracking records collected. IP address of the computer will be recorded to know the demographic location of the participant.

How will my [data/samples/images] be used?
The recordings will be used to examine how one goes about solving the task i.e., using the verbal responses and using the screen recording. They will not be identifying i.e., we will not use specific videos in any form for publication. We will store them on a password protected drive. No video of the person will be recorded. Only the screen will be recorded.

Your eye tracking data and related code will be sent to researchers outside of the University of Nebraska-Lincoln for collaboration with researchers. Any personal information that could identify you will be removed before the data are shared.

In addition, de-identified data will be made available online as a dataset artifact.

What are the possible risks of being in this research study?
There are minimal risks associated with this study such as that of mental distress that could occur during programming. Tasks in this study are timed and you may feel uncomfortable with timed tasks but it is the easiest method to electronically track time in a fair and consistent way.

What are the possible benefits to you?
You may learn about programming languages by participating in this study. Other than possibly contributing to scientific knowledge in the field, there are otherwise no benefits for participation. However, you may not get any benefit from being in this research study.

What are the possible benefits to other people?
The benefits to science and/or society may include better understanding of how to help design new programming languages and understand/improve current ones.

What will being in this research study cost you?
There is no cost to you to be in this research study.

Will you be compensated for being in this research study?
You will receive [redacted] for completing this study. The payment will be in the form of a gift card.

What should you do if you have a problem during this research study?

Your welfare is the major concern of every member of the research team. If you have a problem as a direct result of being in this study, you should immediately contact one of the people listed at the beginning of this consent form.

How will information about you be protected?
Reasonable steps will be taken to protect your privacy and the confidentiality of your study data. The data will be stored electronically through a secure server and will only be seen by the research team during the study and will be archived after the study is complete. The only persons who will have access to your research records are the study personnel, the Institutional Review Board (IRB), and any other person, agency, or sponsor as required by law. The information from this study may be published in scientific journals or presented at scientific meetings but the data will be reported as group or summarized data and your identity will be kept strictly confidential.

What are your rights as a research subject?
You may ask any questions concerning this research and have those questions answered before agreeing to participate in or during the study.

For study related questions, please contact the investigator(s) listed at the beginning of this form.

For questions concerning your rights or complaints about the research contact the Institutional Review Board (IRB):

- Phone: 1(402)472-6965
- Email: irb@unl.edu

What will happen if you decide not to be in this research study or decide to stop participating once you start?
You can decide not to be in this research study, or you can stop being in this research study ("withdraw') at any time before, during, or after the research begins for any reason. Deciding not to be in this research study or deciding to withdraw will not affect your relationship with the investigator or with the University of Nebraska-Lincoln. You will not lose any benefits to which you are entitled.

Documentation of informed consent
You are voluntarily making a decision whether or not to be in this research study. Signing this form means that (1) you have read and understood this consent form, (2) you have had the consent form explained to you, (3) you have had your questions answered and (4) you have decided to be in the research study. You will be given a copy of this consent form to keep.

Participant Feedback Survey
The University of Nebraska-Lincoln wants to know about your research experience. This 14 question, multiple-choice survey is anonymous. This survey should be completed after your participation in this research. Please complete this optional online survey at: http://bit.ly/UNLresearchfeedback.

Participant Name:

_____
(Name of Participant: Please print)

Participant Signature:

_____
Signature of Research Participant

_____
Date

Info

Please write your name and last name.

Please write your email address. We will use this email address to send you an electronic gift card after finishing this study.

Instructions

Thank you for agreeing to be a participant in our study.

In this study, you will be presented a pre-survey, 11 programming questions, two cognitive tasks, and a post-survey.

- You will see 8 questions as part of a pre-survey that will ask you about some demographic information.
- After completing the pre-survey, you will start the study. The first 5 programming questions are to measure your abilities in programming with the C language.
- The first question serves as a tutorial and you will see the answer to this question.
- The next 6 questions will ask you to check if some assertions included in code snippets hold.
- Finally, you will see the links to complete two different cognitive tasks and will be asked to complete them.
- You will be asked to answer 6 questions that will ask you about your experience with this study.

Click on the arrow to start the study.

PreSurvey

The next 8 questions ask you about some demographic information.

What is your age?

- ○ 19-23

- ○ 24-28
- ○ 29-33
- ○ 34-38
- ○ 39-43
- ○ 44-48
- ○ 49+

What is your gender?

- ☐ Woman
- ☐ Man
- ☐ Non-binary
- ☐ Prefer not to disclose
- ☐ Prefer to self-describe:

How do you estimate your programming experience compared to your peers?

- ○ Less experienced than my peers
- ○ Equally experienced to my peers
- ○ More experienced than my peers

How many years of programming experience do you have?

- ○ Less than 1 year
- ○ Between 1 and 3 years
- ○ Between 3 and 5 years
- ○ More than 5 years

Static analysis tools automatically examine source code before a program is run and they can detect defects and bugs. Some examples of static analysis tools are Coverity, FxCop,

Clang, FindBugs, etc. Based on this definition, have you used a static analysis tool before? If so, please indicate what tools you have worked with.

- ○ Yes
- ○ No

How often do you fix bugs?

- ○ A couple times a year
- ○ Every month
- ○ Every week
- ○ Everyday

What IDE are the most you familiar with?

- ○ Visual Studio
- ○ Eclipse
- ○ NetBeans
- ○ Intellij
- ○ Other

How many years have you worked in industry?

- ○ 0-1
- ○ 2-4
- ○ 5+

ProficiencyIntro

Next, you will be asked to answer 5 short programming questions to measure your abilities in programming with the C language.

Proficiency Test

what would be the output if option = 'H' ?

```
switch(option) {
    case 'H' : printf("Hello ");
    case 'W' : printf("Welcome ");
    case 'B' : printf("Bye");
    break;
}
```

- ○ Hello
- ○ Hello Welcome
- ○ Hello Welcome Bye
- ○ None of the above

What will be the output of the following program?

```
#include <stdio.h>
int main(){
    int a = 2;
    int b = 10;
    int y = (b == 0) ? a : (a > b) ? (b = 4) : (b == 0 || 30);
    printf("%d\n", y);
}
```

- ○ 4
- ○ 2
- ○ 1
- ○ Compilation error

What is the output of the following program?

```
#include <stdio.h>
void main(){
    int a;
    a=1;
    while(a<=10){
```

```
        printf("%d ",a);
        if(a>3)
            break;
        a++;
    }
    printf("%d",a+10);
}
```

- 1 2 3 4 13
- 1 2 3 3 14
- 1 2 3 3 13
- 1 2 3 4 14

What is the output of the following program?

```
#include <stdio.h>
void abc(int a){
    ++a;
    printf("%d ", a);
}
void main(){
    int a=10;
    abc(++a);
    abc(a++);
    printf("%d", a);
}
```

- 11 12 12
- 11 12 13
- 12 12 12
- 12 12 13

What is the output of the following program?

```
#include <stdio.h>
void main(){
    int pskills[] = { 10, 20, 30, 40, 50 };
    int i, *ptr ;
    ptr = pskills;
    for (i = 0 ; i <4 ; i++)
    {
       fun(ptr++);
```

```
      printf("%d", *ptr);
   }
}
void fun(int *i){
   *i = *i + 1;
}
```

- 11 21 31 41
- 20 30 40 50
- 21 31 41 51
- 10 20 30 40

TutorialGuide

The next question serves as a tutorial on the assertion tasks. You will see the answer and the explanation of the answer to this question. Please choose the indicated answer to proceed to the next questions.

B-Learning-orig

B-0-A - Inspect the file presented to you in Eclipse. Does the assertion included on line 142 hold?

- Yes
- No

B-0-B - Inspect the file presented to you in Eclipse. Does the assertion included on line 32 hold?

- Yes
- No

Please provide any comments you have:
(NO) The assertion fails (when the number of arguments to main are > 99)

AssertionIntro

The next 6 questions will ask you to examine whether assertions included in the code hold. You can provide your comments as well.

Manual Inspection of Alarms

A-1 - Inspect the file presented to you in Eclipse. Does the assertion included on line 344 hold?

- Yes
- No

A-2 - Inspect the file presented to you in Eclipse. Does the assertion included on line 223 hold?

- Yes
- No

A-3 - Inspect the file presented to you in Eclipse. Does the assertion included on line 363 hold?

- Yes
- No

A-4 - Inspect the file presented to you in Eclipse. Does the assertion included on line 171 hold?

- Yes
- No

B-1

B-1-A - Inspect the file presented to you in Eclipse. Does the assertion included on line 122 hold?

- Yes
- No

B-1-B - Inspect the file presented to you in Eclipse. Does the assertion included on line 40 hold?

- Yes
- No

B-2-A

B-2-A - Inspect the file presented to you in Eclipse. Does the assertion included on line 63 hold?

- Yes
- No

B-2-B

B-2-B - Inspect the file presented to you in Eclipse. Does the assertion included on line 38 hold?

- Yes
- No

C-1

C-1-A - Inspect the file presented to you in Eclipse. Do the assertions added on lines 86, 97, 107, 122, 148 hold?

- Yes
- No

C-1-B - Inspect the file presented to you in Eclipse. Does the assertion added on line 34 hold?

- Yes
- No

C-2-A

C-2-A - Inspect the files presented to you in Eclipse. Do the assertions added in ".\source\Hole_bcs.h" (Lines 195, 216) and ".\source\HMEPbcs.h" (Lines 194, 215) files hold?

- ○ Yes
- ○ No

C-2-B

C-2-B - Inspect the files presented to you in Eclipse. Does the assertion added in file ".\source\updating.h" hold? (Line 65)

- ○ Yes
- ○ No

PsychIntro

The next two questions will guide you to participate in two different cognitive tasks.

PsychTest1

Please use this link to participate in the first cognitive task. The link will open in a new tab. Make sure not to close the current tab/window and return to it after you are finished with the cognitive task. You will be asked to enter the following ID on the first page:

${e://Field/PsychTestID}

https://mrt3dseresllab.netlify.app/

If for any reason you could not open or complete the test, please send an email to niloofar@huskers.unl.edu with the description of the problem.

- ○ Choose this option once you have submitted the the test.
- ○ Choose this option if you could not submit the test.

PsychTest2

Please use this link to participate in the second cognitive task. The link will open in a new tab. Make sure not to close the current tab/window and return to it after you are finished with the

cognitive task. You will be asked to enter the following ID on the first page:

${e://Field/PsychTestID}

https://ospanseresllab.netlify.app

If for any reason you could not open or complete the test, please send an email to niloofar@huskers.unl.edu with the description of the problem.

- ○ Choose this option once you have submitted the the test.
- ○ Choose this option if you could not submit the test.

PostSurvey

The next 6 questions are short questions asking you about your experience doing this survey.

Did you have sufficient time to complete the tasks?

- ○ Yes
- ○ No

How would you rate the overall difficulty of the tasks?

- ○ Extremely difficult
- ○ Somewhat difficult
- ○ Neither easy nor difficult
- ○ Somewhat easy
- ○ Extremely easy

Rate your programming skills.

- ○ Extremely good
- ○ Somewhat good
- ○ Neither good nor bad
- ○ Poor
- ○ Extremely poor

Rate your design skills.

- ○ Extremely good
- ○ Somewhat good
- ○ Neither good nor bad
- ○ Poor
- ○ Extremely poor

Describe any difficulty you faced during the study.

Any other comments about the study are welcome.

Final

Thank you for participating in our study.

Please click the next button (arrow button) to submit your survey. The survey will NOT be submitted if you don't click the button.

B.2 Study Overview

Assessment of Static Analysis Alarms: An Eye Tracking Study

Dear participant, Thank you for agreeing to participate in our study. In this eye tracking study, you will be asked to sit in front of a computer monitor and solve several C programming tasks and two cognitive tasks. You will also be asked to complete a pre and post survey. After completing each task, you will be asked to verbally explain your rationale for solving that task and your answer will be recorded by the moderator.

First, we will calibrate the eye tracking system for use. You will need to follow the dot that moves around on the screen with your eyes. It is important that you do not make a lot of movements during the study, do not change your position on the chair, and do not change the chair placement either. The moderator will let you know if you have moved too much and how you can correct your position.

You will see overall instructions and instructions for individual tasks on a webpage within a Qualtrics survey. The moderator will also explain the instructions to you. You will see all the C programming tasks on the Eclipse IDE. You are not allowed to run the code or change it. After thinking about the tasks and solving them you will have to say, "I'm done with this task". After completing each task, you will be guided back to the Qualtrics page to specify your answer, and then you will answer some questions that the moderator asks from you.

You can take as long as you want working on each task. The moderator will not be able to answer any questions related to the tasks.

Some Important Notes:

- Do not move or resize any windows.

- You can use ctrl+F to search in code, but you will need to move the search window to the side.
- Do not hit the Escape key.
- Code folding should be left off.
- Do not change the font size.
- Please avoid using a print statement for debugging.
- Refrain from using the F1 key.
- The moderator will keep you informed of when to start your tasks.
- You may not use the web to find answers.

B.3 Relevant Blocks of Code For Tasks in Group A: Manual Inspection of Alarms

Tasks of group A each have an assertion on one single line. The assertions are not repositioned.

Task A-1.

```
      1  /*
      2  Does the assertion included on line 344 hold?
      3  */
Block 2  ...
    224  char *
    225  read_passphrase(const char *prompt, int flags)
    226  {
Block 3  ...
    250    if ((flags & RP_USE_ASKPASS) && getenv("DISPLAY") == NULL)
    251      return (flags & RP_ALLOW_EOF) ? NULL : xstrdup("");
    252
    253    if (use_askpass && getenv("DISPLAY")) {
    254      if (getenv(SSH_ASKPASS_ENV))
    255        askpass = getenv(SSH_ASKPASS_ENV);
    256      else
    257        askpass = _PATH_SSH_ASKPASS_DEFAULT;
    258      if ((ret = ssh_askpass(askpass, prompt)) == NULL)
    259        if (!(flags & RP_ALLOW_EOF))
    260                  return xstrdup("");
    261        return ret;
    262    }
    263
    264    if ((flags & RP_USE_ASKPASS) &&
    265      readpassphrase(prompt, buf, sizeof buf, rppflags) == NULL) {
    266      return NULL;
    267    }
    268
    269    ret = xstrdup(buf);
    270    memset(buf, 'x', sizeof buf);
    271    return ret;
    272  }
Block 4  ...
    274  static char *
    275  ssh_askpass(char *askpass, const char *msg)
    276  {
Block 5  ...
    284    if (fflush(stdout) != 0)
    285      error("ssh_askpass: fflush: %s", strerror(errno));
    286    if (askpass == NULL)
```

```
     287      fatal("internal error: askpass undefined");
     288    if (pipe(p) < 0) {
     289      error("ssh_askpass: pipe: %s", strerror(errno));
     290      return NULL;
     291    }
     292    osigchld = signal(SIGCHLD, SIG_DFL);
     293    if ((pid = fork()) < 0) {
     294      error("ssh_askpass: fork: %s", strerror(errno));
     295      signal(SIGCHLD, osigchld);
     296      return NULL;
     297    }
Block 6  ...
     325    if (ret == -1 || !WIFEXITED(status) || WEXITSTATUS(status) != 0) {
     326      memset(buf, 0, sizeof(buf));
     327      return NULL;
     328    }
     329
     330    buf[strcspn(buf, "\r\n")] = '\0';
     331    pass = xstrdup(buf);
     332    memset(buf, 0, sizeof(buf));
     333    return pass;
     334  }
Block 1  ...
     336  static int
     337  confirm(const char *prompt)
     338  {
     339    char *p;
     340    int ret = -1;
     341    const char* msg = prompt;
     342    if(!msg) return 0;
     343    p = read_passphrase(msg, RP_ECHO);
     344    assert(p!=NULL);
```

Listing B.1: The relevant blocks for task A-1.
Block numbers for each block are specified at the beginning of each block.

Task A-2.

```
       1  /*
       2  Does the assertion included on line 223 hold?
       3  */
Block 4   ...
     181    char *askpass = NULL, *ret = NULL, buf[1024];
     182    int t;
     183    int rppflags, use_askpass = 0, ttyfd;
Block 3   ...
     204    t = flags & RP_USE_ASKPASS;
     205    if ((flags & RP_USE_ASKPASS) || !(ret = getenv("DISPLAY")))
     206    { t = 2;
     207      goto end;
     208    }
Block 2   ...
     210    if (use_askpass && getenv("DISPLAY")) {
     211      if (getenv(SSH_ASKPASS_ENV))
     212        askpass = getenv(SSH_ASKPASS_ENV);
     213      else
     214        askpass = _PATH_SSH_ASKPASS_DEFAULT;
     215      if((ret = ssh_askpass(askpass, prompt)) == NULL)
     216                goto end;
     217    }
Block 1   ...
     220    ret = buf;
     221    memset(buf, 'x', sizeof buf);
     222    end:
     223    assert(ret != NULL);
     224    return ret;
     225  }
```

Listing B.2: The relevant blocks for task A-2.
Block numbers for each block are specified at the beginning of each block.

Task A-3.

```
      1   /*
      2   Does the assertion included on line 363 hold?
      3   */
Block 2   ...
     88   struct {
     89     Key        *server_key;        /* ephemeral server key */
     90     Key        *ssh1_host_key;        /* ssh1 host key */
     91     Key        **host_keys;        /* all private host keys */
     92     int        have_ssh1_key;
     93     int        have_ssh2_key;
     94     u_char        ssh1_cookie[SSH_SESSION_KEY_LENGTH];
     95   } sensitive_data;
Block 5   ...
    170   /* Destroy the host and server keys.  They will no longer be needed. */
    171   void
    172   destroy_sensitive_data(void)
Block 3   ...
    180     for (i = 0; i < options.num_host_key_files; i++) {
    181       if (sensitive_data.host_keys[i]) {
    182         key_free(sensitive_data.host_keys[i]);
    183         sensitive_data.host_keys[i] = NULL;
Block 4   ...
    339     /* Destroy the private and public keys. No longer. */
    340     destroy_sensitive_data();
Block 1   ...
    362     for (i = 0; i < options.num_host_key_files; i++) {
    363       assert(sensitive_data.host_keys[i] == NULL);
    364     }
```

Listing B.3: The relevant blocks for task A-3.
Block numbers for each block are specified at the beginning of each block.

Task A-4.

```
       1  /*
       2  Does the assertion included on line 171 hold?
       3  */
       4  unsigned int optind = 0;
Block 4   ...
      75  /* Reset this to zero so that getopt internals get initialized from
      76  the probably-new parameters when/if getopt is called later.  */
      77  optind = 0;
      78  }
Block 7   ...
      87  int
      88  getopt_long (int argc, char **argv,  char *options,
      89  struct option *long_options, int *opt_index)
      90  {
      91    if(argc < 1) return -1;
      92    if(optind == 0) optind = 1;
      93
      94    while( skip(argv[optind]) && optind<argc)
      95    {
      96      optind++;
      97    }
      98
      99    if(optind>=argc) return -1;
     100    optind++;
     101    if(str_prefix(options, argv[optind]))
     102    {
     103      optarg = argv[optind];
     104      return 0;
     105    }
     106    return -1;
     107  }
Block 5   ...
     141    parse_long_options (argc, argv, PROGRAM_NAME, PACKAGE_NAME, Version,usage);
Block 6   ...
     143    if (getopt_long (argc, argv, "+", NULL, NULL) != -1){
     144      usage (EXIT_FAILURE);
     145    }
Block 2   ...
     148    if (argc <= optind)
     149    {
     150      error (0, 0, "missing operand");
     151      usage (EXIT_FAILURE);
     152    }
Block 3   ...
     158    if (argc == optind + 1)
     159    {
     160      char *shell = getenv ("SHELL");
     161      if (shell == NULL)
     162        shell = bad_cast ("/bin/sh");
```

```
     163      argv[0] = shell;
     164    }
     165    else
     166    {
     167      /* The following arguments give the command.  */
     168      argv += optind + 1;
Block 1      ...
     171    assert(optind >=2);
```

Listing B.4: The relevant blocks for task A-4.
Block numbers for each block are specified at the beginning of each block.

B.4 Relevant Blocks of Code For Tasks in Group B: Manual Inspection of Repositioned Alarms

Task B-1.

Task B-1 was presented to the participants either without treatment (B-1-A) or with treatment (B-1-B). The assertion for the task without the treatment is specified on line 122. The version with the treatment had the repositioned alarm on line 40. Note that the participants received either the task without the treatment, with the original alarm, or with the treatment with the repositioned alarm.

```
 1  /*
 2  Does the assertion included on line 122 (line 40) hold?
 3  */
 4
 5  #define NULL 0
 6  #define MAX_BUFLEN 1024
 7  int buffer[64],index,acpid_debug;
Block 2
...
35          index = 0;
36          buflen = handle_cmdline(do_detach);
37          while (1){
38                  if (do_detach) {
39                          /* tell our clients to buzz off */
40                          assert(index >= 0 && index <= 63);(Repositioned Alarm)
41                          acpid_add_client(do_detach,p->origin);
42                          p = client_list.head;
43                          while (p) {
44                                  next = p->next;
45                                  close(p->action.fd);
46                                  p = next;
47                          }
48                  }
49                  index ++;
50                  if (buflen >= MAX_BUFLEN) {
51                          break;
52                  }
53                  buflen *= 2;
54          }
Block 4
...
64  int
65  acpid_add_client(int clifd, const char *origin)
66  {
67          struct rule *r;
```

```
        int nrules = 0;

        r = parse_client(clifd);
        if (r) {
                r->origin = strdup(origin);
                nrules++;
        }
        return 0;
}
Block 3 ...
int handle_cmdline (int do_detach){
        if (!do_detach) {
                printf("ERR: malloc(%d): %s\n");
        }

        if(do_detach >= MAX_BUFLEN || do_detach < 0)
                return MAX_BUFLEN - 1;
        else
                return do_detach;
}
Block 6 ...
static void
enlist_rule(struct rule_list *list, struct rule *r)
{
        if (!list->head) {
                list->head = r;
                list->tail = r;
        } else {
                list->tail->next = r;
                list->tail = r;
        }
        do_client_rule(r,r->origin);
}
Block 5 ...
static struct rule *
parse_client(int client)
{
        struct rule *r;
        int rv;

        if (rv) {
                char buf[128];
                regerror(rv, r->event, buf, sizeof(buf));
        }
        enlist_rule(&client_list, r);
        return r;
}
Block 1 ...
static void
unlock_rules(void)
{
```

```
119         if (acpid_debug >= 4) {
120                 acpid_log("unblocking signals for rule lock\n");
121         }
122         assert(index >= 0 && index <= 63); (Original Alarm)
123         acpid_log(buffer[index]);
124 }
Block 7 ...
126 static int
127 do_client_rule(struct rule *rule, const char *event)
128 {
129         int r;
130         int client = rule->action.fd;
131
132         if (r < 0 ) {
133                 /* closed */
134                 close(rule->action.fd);
135                 free_rule(rule);
136         }
137         unlock_rules();
138         return 0;
139 }
```

Listing B.5: The relevant blocks for task B-1.
Block numbers for each block are specified at the beginning of each block.

Task B-2

Task B-2 was presented to the participants either without treatment (B-2-A) or with treatment (B-2-B). The assertion for the task without the treatment is specified on line 63. The version with the treatment had the repositioned alarm on line 38. Note that the participants received either the task without the treatment, with the original alarm, or with the treatment with the repositioned alarm.

```
      1  /*
      2  Does the assertion included on line 63 (line 38) hold?
      3  */
Block 5  ...
     26  long main1(int argc,char **argv[],RingBuffer * rbuf,long numBytes,void *dataPtr)
     27  {//RingBuffer_Init
     28      if (rbuf != NULL) {
     29          for (i = 0; i < argc; i++) {
     30              if (strcmp("piecewise_linear", argv[i]) == 0) {
     31                  fid = i;
     32                  break;
     33              }
     34          }
     35      }
     36      index = fid;
     37      if(dataPtr != NULL){
     38              assert(index >= -2 && index <= 97); (Repositioned Alarm)
     39              RingBuffer_Read(rbuf, dataPtr, numBytes);
Block 1  ...
     55  long
     56  RingBuffer_GetWriteRegions(RingBuffer * rbuf, long numBytes,
     57                             void **dataPtr1, long *sizePtr1,
     58                             void **dataPtr2, long *sizePtr2)
     59  {
     60      long tindex;
     61      long available;
     62      index ++;
     63      assert(index >= 0 && index <= 99); (Original Alarm)
     64      available = buf[index];
Block 3  ...
     92  long
     93  RingBuffer_GetReadRegions(RingBuffer * rbuf, long numBytes,
     94                            void **dataPtr1, long *sizePtr1,
     95                            void **dataPtr2, long *sizePtr2)
     96  {
     97      long tindex,available;
     98      index--;
     99      RingBuffer_Write(rbuf, sizePtr1, numBytes);
```

```
Block 2
     121  long
     122  RingBuffer_Write(RingBuffer * rbuf, void *data, long numBytes)
     123  {
     124      long size1, size2, numWritten;
     125      void *data1, *data2;
     126      index = index + 2;
     127      numWritten =
     128          RingBuffer_GetWriteRegions(rbuf, numBytes, &data1, &size1, &data2,
     129                                     &size2);
Block 4
     145  long
     146  RingBuffer_Read(RingBuffer * rbuf, void *data, long numBytes)
     147  {
     148      long size1, size2, numRead;
     149      void *data1, *data2;
     150      numRead =
     151          RingBuffer_GetReadRegions(rbuf, numBytes, &data1, &size1, &data2,
     152                                    &size2);
```

Listing B.6: The relevant blocks for task B-2.
Block numbers for each block are specified at the beginning of each block.

B.5 Relevant Blocks of Code For Tasks in Group C: Manual Inspection of Merged and Repositioned Alarms

Task C-1

Task C-1 was presented to the participants either without treatment (C-1-A) or with treatment (C-1-B). The assertions for the task without the treatment are specified on lines 86, 97, 107, 122 and 148. The version with the treatment had the merged and repositioned alarm on line 34. Note that the participants received either the task without the treatment, with the original alarms, or with the treatment with the repositioned alarm.

```
      1  /*
      2  Do the assertions added to lines 86, 97, 107, 122, 148 (line 34) hold?
      3  */
Block 6  ...
     17  int main1(int argc, char ** argv, FILE *pFile, long lFilesize)
     18  {
     19          int        iGuess,iFirst,szTask, iWordVersion;
     20          if (argc <= 0) {
     21                  return EXIT_FAILURE;
     22          }
     23
     24          szTask = szBasename();
     25
     26          if (argc <= 1) {
     27                  iFirst = 1;
     28                  bUsage = TRUE;
     29          } else {
     30                  iFirst = iReadOptions(argc, argv);
     31          }
     32
     33          for (index = iFirst; index <= argc; index++) {
     34                  assert(index >= 0 && index <= 99); (Repositioned Alarm)
     35                  iGuess = nondet();
     36                  switch (iGuess) {
     37                  case 0:
     38                          bIsWordForDosFile(pFile, lFilesize);
     39                          break;
     40                  case 2:
     41                          bIsWinWord12File(pFile, lFilesize);
     42                          break;
     43                  case 5:
```

```
44                              bIsMacWord45File(pFile);
45                              break;
46                      case 6:
47                              bIsRtfFile(pFile);
48                              break;
49                      default:
50                              bIsWordPerfectFile(pFile);
51                              iWordVersion = -1;
52                              break;
53                      }
54              }
55
56              return iWordVersion;
57      } /* end of iInitDocument */
58
Block 1 ...
78      int bIsWordForDosFile(FILE *pFile, long lFilesize)
79      {
80              static char aucBytes[] ={ 0x31, 0xbe, 0x00, 0x00, 0x00, 0xab};
81              DBG_MSG("bIsWordForDosFile");
82
83              if (pFile == NULL || lFilesize < 0) {
84                      DBG_MSG("No proper file given");
85              }
86              assert(index >= 0 && index <= 99); (Original Alarm)
87              size = Bytes[index];
Block 2 ...
92      int bIsRtfFile(FILE *pFile)
93      {
94              static char        aucBytes[] ={ '{', '\\', 'r', 't', 'f', '1' };
95
96              DBG_MSG("bIsRtfFile");
97              assert(index >= 0 && index <= 99); (Original Alarm)
98              size = Bytes[index];
Block 3 ...
102     int bIsWordPerfectFile(FILE *pFile)
103     {
104             static char        aucBytes[] ={ 0xff, 'W', 'P', 'C' };
105
106             DBG_MSG("bIsWordPerfectFile");
107             assert(index >= 0 && index <= 99); (Original Alarm)
108             size = Bytes[index];
Block 4 ...
113     int bIsWinWord12File(FILE *pFile, long lFilesize)
114     {
115             static char        aucBytes[2][4] = {
116                     { 0x9b, 0xa5, 0x21, 0x00 },        /* Win Word 1.x */
117                     { 0xdb, 0xa5, 0x2d, 0x00 },        /* Win Word 2.0 */
118             };
119             int        iIndex;
```

```
120         DBG_MSG("bIsWinWord12File");
121
122         assert(index >= 0 && index <= 99); (Original Alarm)
123         size = Bytes[index];
Block 5  ...
139  int bIsMacWord45File(FILE *pFile)
140  {
141         static char        aucBytes[2][6] = {
142                 { 0xfe, 0x37, 0x00, 0x1c, 0x00, 0x00 },        /* Mac Word 4 */
143                 { 0xfe, 0x37, 0x00, 0x23, 0x00, 0x00 },        /* Mac Word 5 */
144         };
145         int        iIndex;
146         DBG_MSG("bIsMacWord45File");
147
148         assert(index >= 0 && index <= 99); (Original Alarm)
149         size = Bytes[index];
Block 7  ...
184  int iReadOptions(int argc, char ** argv){
185         int iIndex;
186         for (iIndex = 0; iIndex < argc; iIndex++) {
187                 if(bProcessFile(argv[iIndex])){
188                         return iIndex;
189                 }
190         }
```

Listing B.7: The relevant blocks for task C-1.
Block numbers for each block are specified at the beginning of each block.

C-2

Task C-2 was presented to the participants either with or without treatment. The assertions for the task are without treatment are specified on lines 195 and 216 of the file *Hole_bcs.h* and on lines 194 and 215 of the file *HMEPbcs.h*. The version with the treatment had the repositioned alarm on line 65 of the file *updating.h*. The participants had access to the entire project folder for this task, which contained 38 files. We have added the full code of the relevant files for solving the tasks, and have specified the important blocks in each program. Note that the participants received either the task without the treatment, with the original alarm, or with the treatment with the repositioned alarm.

Task C-2-A.

Hole_bcs.h

```
          1   /*
          2   Do the assertions added in ".\source\Hole_bcs.h" (line 195, 216) and
          3   ".\source\HMEPbcs.h" (line 194, 215) files hold?
          4   */
Block3Start   ...
         41   void
         42   HoleHMEPBCs(void)
         43   {
         44    int i,j;
         45    real xvel,yvel;
  Block3End   ...
Block1Start      for(i=1;i<=nx+2;i++){
        190   // INSULATOR
        191        if(EDGE[2][i][0]==0){
        192   // density
        193          h2d[i+2][ny+2][1]=h2d[i+2][ny+1][1];
        194          h2d[i+2][ny+3][1]=h2d[i+2][ny][1];
        195          assert((ny+4)>=0 && (ny+4)<=308); (Original Alarm)
        196          h2d[i+2][ny+4][1]=h2d[i+2][ny-1][1];
  Block1End   ...
Block2Start   // density
        214          h2d[i+2][ny+2][1]=EDGE[2][i][3];
        215          h2d[i+2][ny+3][1]=EDGE[2][i][3];
        216                       assert((ny+4)>=0 && (ny+4)<=308); (Original Alarm)
```

```
Block2End         h2d[i+2][ny+4][1]=EDGE[2][i][3];
```

Holemep2d.h

```
Block2Start
          44   void
          45   Hole_MEP2D(int nx,int ny,real dx,
          46              real dy,real cfl,real theta)
          47   {
  Block2End    ...block1
Block1Start    // Start a 2-stage time iteration
          85    for(io=0;io<=1;io++){
  Block1End       HoleHMEPBCs();
```

HMEPbcs.h

```
           1   /*
           2   Do the assertions added in ".\source\Hole_bcs.h" (line 195, 216) and
           3   ".\source\HMEPbcs.h" (line 194, 215) files hold?
           4   */
           5   ...
Block3Start    void
          41   HMEPBCs(void)
          42   {
          43    int i,j;
          44    real xvel,yvel;
  Block3End    ....
Block1Start       for(i=1;i<=nx+2;i++){
         189   // INSULATOR
         190        if(EDGE[2][i][0]==0){
         191   // density
         192          u2d[i+2][ny+2][1]=u2d[i+2][ny+1][1];
         193          u2d[i+2][ny+3][1]=u2d[i+2][ny][1];
         194          assert((ny+4)>=0 && (ny+4)<=308); (Original Alarm)
         195          u2d[i+2][ny+4][1]=u2d[i+2][ny-1][1];
  Block1End    ...
Block2Start    // density
         213          u2d[i+2][ny+2][1]=EDGE[2][i][2];
         214          u2d[i+2][ny+3][1]=EDGE[2][i][2];
         215          assert((ny+4)>=0 && (ny+4)<=308); (Original Alarm)
  Block2End          u2d[i+2][ny+4][1]=EDGE[2][i][2];
```

ParabMEP2D.h

```
Block2Start    void ParabMEP2D(int nx,int ny,real dx,real dy,real cfl,real theta)
          58   {
```

```
  Block2End    ...
Block1Start    // Start a 2-stage time iteration
          93    for(io=0;io<=1;io++){
          94      HMEPBCs();
  Block1End    ...
```

updating.h

```
Block3Start    void
          39   updating(int model)
          40   {
  Block3End    ...
Block1Start      if(model==MEPE  model==MEPEH){
          61       DT/=2.;
          62       ParabMEP2D(nx,ny,dx,dy,0.475,1.0);
  Block1End        electron_relaxation_step();
Block2Start    ...
          69     if(model==MEPH  model==MEPEH){
          70       DT/=2.;
  Block2End        Hole_MEP2D(nx,ny,dx,dy,0.475,1.0);
```

archimedes.c

```
Block3Start    #define MN3 4
          51   #define NYM 308
          52   #define DIME 1003
  Block3End    ...
Block2Start
         485   // Read the geometrical and physical description of the MESFET
         486   // ========================================================
         487        Read_Input_File();
  Block2End    ...
Block1Start    // ======================
         695        for(c=1;c<=ITMAX;c++) updating(Model_Number);
         696   // Here we save the outputs
  Block1End    ...
```

readinputfile.h

```
Block1Start    Model_Number=MCE; // here we choose the model
          53   nx=50; // default # of cells in x-direction
```

```
          54    ny=50; // default # of cells in y-direction
          55    TF=5.0e-12; // Default Final Time in seconds
    Block1End   ...
  Block2Start     else if(strcmp(s,"YSPATIALSTEP")==0){
         275        fscanf(fp,"%lf",&num);
         276        ny=(int) num;
         277        if(ny>NYM){
         278          printf("%s: too large y-spatial step\n",progname);
         279          exit(EXIT_FAILURE);
         280        }
         281        if(LYflag==0){
         282          printf("%s: you have to define the y-length first\n",progname);
         283          exit(EXIT_FAILURE);
         284        }
         285        dy=LY/ny; // spatial step in y-direction
         286        printf("YSPATIALSTEP = %d ---> Ok\n",ny);
    Block2End     }
```

Task C-2-B.

updating.h

```
1   /*
2   Does the assertion included on line 65 hold?
3   */
4   ...
Block3Start
43  void
44  updating(int model)
    {
Block3End
    ...
Block1Start
      if(model==MEPE  model==MEPEH){
65      assert((ny+4)>=0 && (ny+4)<=308); (Merged and Repositioned Alarm)
66      DT/=2.;
67      ParabMEP2D(nx,ny,dx,dy,0.475,1.0);
68      electron_relaxation_step();
Block1End
    ...
Block2Start
      if(model==MEPH  model==MEPEH){
74      DT/=2.;
75      Hole_MEP2D(nx,ny,dx,dy,0.475,1.0);
Block2End
        Relaxation_Step_Hole();
```

archimedes.c

```
1   #define MN3 4
2   #define NYM 308
3   #define DIME 1003
4   ...
5
6   // Read the geometrical and physical description of the MESFET
7   // ============================================================
8        Read_Input_File();
9   ...
10  // ======================
11       for(c=1;c<=ITMAX;c++) updating(Model_Number);
12  // Here we save the outputs
13  ...
```

readinputfile.h

```
Block1Start
          Model_Number=MCE; // here we choose the model
53        nx=50; // default # of cells in x-direction
```

```
          54    ny=50; // default # of cells in y-direction
          55    TF=5.0e-12; // Default Final Time in seconds
   Block1End   ...
  Block2Start    else if(strcmp(s,"YSPATIALSTEP")==0){
         275       fscanf(fp,"%lf",&num);
         276       ny=(int) num;
         277       if(ny>NYM){
         278         printf("%s: too large y-spatial step\n",progname);
         279         exit(EXIT_FAILURE);
         280       }
         281       if(LYflag==0){
         282         printf("%s: you have to define the y-length first\n",progname);
         283         exit(EXIT_FAILURE);
         284       }
         285       dy=LY/ny; // spatial step in y-direction
         286       printf("YSPATIALSTEP = %d ---> Ok\n",ny);
    Block2End    }
```

Bibliography

[1] N. Mansoor, T. Muske, A. Serebrenik, and B. Sharif, "An empirical assessment on merging and repositioning of static analysis alarms," in *International Working Conference on Source Code Analysis and Manipulation (SCAM)*, 2022. (document)

[2] N. Mansoor, "Empirical assessment of program comprehension styles in programming language paradigms," in *2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–2, IEEE, 2021. (document)

[3] B. Sharif and N. Mansoor, "Humans in empirical software engineering studies: An experience report," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pp. 1286–1292, IEEE, 2022. (document)

[4] D. Hovemeyer and W. Pugh, "Finding bugs is easy," *SIGPLAN Not.*, vol. 39, p. 92–106, dec 2004. 1

[5] N. Rutar, C. B. Almazan, and J. S. Foster, "A comparison of bug finding tools for java," in *15th International symposium on software reliability engineering*, pp. 245–256, IEEE, 2004. 1

[6] M. A. Jamil, M. Arif, N. S. A. Abubakar, and A. Ahmad, "Software testing techniques: A literature review," in *2016 6th international conference on information*

and communication technology for the Muslim world (ICT4M), pp. 177–182, IEEE, 2016. 1

[7] V. Garousi and M. V. Mäntylä, "A systematic literature review of literature reviews in software testing," *Information and Software Technology*, vol. 80, pp. 195–216, 2016. 1

[8] N. Ayewah, W. Pugh, D. Hovemeyer, J. D. Morgenthaler, and J. Penix, "Using static analysis to find bugs," *IEEE software*, vol. 25, no. 5, pp. 22–29, 2008. 1

[9] P. Emanuelsson and U. Nilsson, "A comparative study of industrial static analysis tools," *Electronic notes in theoretical computer science*, vol. 217, pp. 5–21, 2008. 1

[10] A. Arusoaie, S. Ciobâca, V. Craciun, D. Gavrilut, and D. Lucanu, "A comparison of open-source static analysis tools for vulnerability detection in c/c++ code," in *2017 19th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 161–168, IEEE, 2017. 1

[11] A. Fatima, S. Bibi, and R. Hanif, "Comparative study on static code analysis tools for c/c++," in *2018 15th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*, pp. 465–469, IEEE, 2018. 1

[12] 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. 1, 3.1

[13] V. D'silva, D. Kroening, and G. Weissenbacher, "A survey of automated techniques for formal software verification," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1165–1178, 2008. 1

[14] E. W. Dijkstra *et al.*, "Notes on structured programming," 1970. 1

[15] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 672–681, 2013. 1, 1, 4.2, 5.2.1

[16] S. Krishnamurthi and T. Nelson, "The human in formal methods," in *Formal Methods – The Next 30 Years* (M. H. ter Beek, A. McIver, and J. N. Oliveira, eds.), (Cham), pp. 3–10, Springer International Publishing, 2019. 1, 2.1

[17] W. Maalej, R. Tiarks, T. Roehm, and R. Koschke, "On the comprehension of program comprehension," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 23, no. 4, pp. 1–37, 2014. 1

[18] S. Panichella, A. Panichella, M. Beller, A. Zaidman, and H. C. Gall, "The impact of test case summaries on bug fixing performance: An empirical investigation," in *Proceedings of the 38th International Conference on Software Engineering*, ICSE '16, (New York, NY, USA), p. 547–558, Association for Computing Machinery, 2016. 1

[19] M. Tufano, G. Bavota, D. Poshyvanyk, M. Di Penta, R. Oliveto, and A. De Lucia, "An empirical study on developer-related factors characterizing fix-inducing commits," *Journal of Software: Evolution and Process*, vol. 29, no. 1, p. e1797, 2017. 1

[20] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *2012 19th Working Conference on Reverse Engineering*, pp. 225–234, 2012. 1

[21] H. Wu, L. Shi, C. Chen, Q. Wang, and B. Boehm, "Maintenance effort estimation for open source software: A systematic literature review," in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 32–43, 2016. 1

[22] R. Stein and S. E. Brennan, "Another person's eye gaze as a cue in solving programming problems," in *Proceedings of the 6th international conference on Multimodal interfaces*, pp. 9–15, 2004. 1, 4.2

[23] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," ETRA '06, (New York, NY, USA), p. 133–140, Association for Computing Machinery, 2006. 1, 4.2

[24] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, (New York, NY, USA), p. 381–384, Association for Computing Machinery, 2012. 1, 4.2

[25] R. Bednarik, "Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations," *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 143–155, 2012. 1, 4.2

[26] R. Turner, M. Falcone, B. Sharif, and A. Lazar, "An eye-tracking study assessing the comprehension of c++ and python source code," in *Proceedings of the*

Symposium on Eye Tracking Research and Applications, pp. 231–234, ACM, 2014. 1, 4.2

[27] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz, "Tracing software developers' eyes and interactions for change tasks," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pp. 202–213, 2015. 1, 4.2

[28] S. Aljehane, B. Sharif, and J. Maletic, "Determining differences in reading behavior between experts and novices by investigating eye movement on source code constructs during a bug fixing task," in *ACM Symposium on Eye Tracking Research and Applications*, pp. 1–6, 2021. 1, 4.2

[29] R. Bednarik and M. Tukiainen, "An eye-tracking methodology for characterizing program comprehension processes," in *Proceedings of the 2006 symposium on Eye tracking research & applications*, pp. 125–132, 2006. 1

[30] T. Busjahn, C. Schulte, and A. Busjahn, "Analysis of code reading to gain more insight in program comprehension," in *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, pp. 1–9, 2011. 1

[31] 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. 1, 2.1, 2.4.3, 3.3.1, 3.4.2.1

[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. 1, 2.1, 2.4.3, 3.3.1, 3.4.2.1

[33] M. C. Corballis, "Mental rotation and the right hemisphere," *Brain and language*, vol. 57, no. 1, pp. 100–121, 1997. 1

[34] M. C. Linn and A. C. Petersen, "Emergence and characterization of sex differences in spatial ability: A meta-analysis," *Child development*, pp. 1479–1498, 1985. 1

[35] R. N. Shepard and J. Metzler, "Mental rotation of three-dimensional objects," *Science*, vol. 171, no. 3972, pp. 701–703, 1971. 1

[36] M. Hegarty and M. Kozhevnikov, "Types of visual–spatial representations and mathematical problem solving.," *Journal of educational psychology*, vol. 91, no. 4, p. 684, 1999. 1, 2.1, 2.7

[37] J. Wai, D. Lubinski, and C. P. Benbow, "Spatial ability for stem domains: Aligning over 50 years of cumulative psychological knowledge solidifies its importance.," *Journal of educational Psychology*, vol. 101, no. 4, p. 817, 2009. 1

[38] M. S. Khine, "Spatial cognition: Key to stem success," in *Visual-spatial ability in STEM education*, pp. 3–8, Springer, 2017. 1, 4.2

[39] M. Daneman and P. A. Carpenter, "Individual differences in working memory and reading," *Journal of verbal learning and verbal behavior*, vol. 19, no. 4, pp. 450–466, 1980. 1

[40] K. P. Raghubar, M. A. Barnes, and S. A. Hecht, "Working memory and mathematics: A review of developmental, individual difference, and cognitive approaches," *Learning and individual differences*, vol. 20, no. 2, pp. 110–122, 2010. 1, 2.1

[41] P. Peng, J. Namkung, M. Barnes, and C. Sun, "A meta-analysis of mathematics and working memory: Moderating effects of working memory domain, type of mathematics skill, and sample characteristics.," *Journal of Educational Psychology*, vol. 108, no. 4, p. 455, 2016. 1

[42] R. Krueger, Y. Huang, X. Liu, T. Santander, W. Weimer, and K. Leach, "Neurological divide: an fmri study of prose and code writing," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, pp. 678–690, IEEE, 2020. 1

[43] G. R. Bergersen and J.-E. Gustafsson, "Programming skill, knowledge, and working memory among professional software developers from an investment theory perspective.," *Journal of individual Differences*, vol. 32, no. 4, p. 201, 2011. 1

[44] 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. 1, 2.1, 2.4.3, 3.1, 3.3.1, 3.4.2.1, 4.2

[45] 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. 1, 2.1, 2.4.3, 3.3.1, 3.4.2.1, 4.2

[46] D. Jackson, *Software Abstractions: logic, language, and analysis*. 2012. 1, 2.1, 2.4.3, 2.4.4

[47] D. Jackson, "How does the Alloy Analyzer differ from model checkers?." 1

[48] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, p. 1, 2002. 1

[49] T. Muske, R. Talluri, and A. Serebrenik, "Repositioning of static analysis alarms," in *Proceedings of the 27th ACM SIGSOFT international symposium on software testing and analysis*, pp. 187–197, 2018. 1

[50] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, "itrace: Eye tracking infrastructure for development environments," in *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, pp. 1–3, 2018. 1, 4.4.1, 4.4.3, 4.5.1

[51] D. Jackson, "Alloy analyzer." `https://alloytools.org/download.html`. 2.1

[52] N. Mirzaei, J. Garcia, H. Bagheri, A. Sadeghi, and S. Malek, "Reducing combinatorics in gui testing of android applications," pp. 559–570, 2016. 2.1

[53] H. Bagheri, A. Sadeghi, J. Garcia, and S. Malek, "Covert: Compositional analysis of android inter-app permission leakage," *IEEE transactions on Software Engineering*, vol. 41, no. 9, pp. 866–886, 2015. 2.1

[54] H. Bagheri, E. Kang, S. Malek, and D. Jackson, "A formal approach for detection of security flaws in the android permission system," *Formal Aspects of Computing*, vol. 30, no. 5, pp. 525–544, 2018. 2.1

[55] M. Alhanahnah, C. Stevens, and H. Bagheri, "Scalable analysis of interaction threats in iot systems," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 272–285, 2020. 2.1

[56] J. P. Near, A. Milicevic, E. Kang, and D. Jackson, "A lightweight code analysis and its role in evaluation of a dependability case," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 31–40, 2011. 2.1, 2.7

[57] N. Mansoor, J. A. Saddler, B. Silva, H. Bagheri, M. B. Cohen, and S. Farritor, "Modeling and testing a family of surgical robots: an experience report," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 785–790, 2018. 2.1, 2.7

[58] M. Spichkova and A. Zamansky, "Teaching of formal methods for software engineering.," in *ENASE*, pp. 370–376, 2016. 2.1

[59] N. Danas, T. Nelson, L. Harrison, S. Krishnamurthi, and D. J. Dougherty, "User studies of principled model finder output," in *International Conference on Software Engineering and Formal Methods*, pp. 168–184, Springer, 2017. 2.1, 2.3.1, 2.7

[60] X. Li, D. Shannon, J. Walker, S. Khurshid, and D. Marinov, "Analyzing the uses of a software modeling tool," *Electronic Notes in Theoretical Computer Science*, vol. 164, no. 2, pp. 3–18, 2006. 2.1, 2.3.1, 2.7, 5.1.1

[61] N. J. Abid, J. I. Maletic, and B. Sharif, "Using developer eye movements to externalize the mental model used in code summarization tasks," in *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pp. 1–9, 2019. 2.1

[62] R. Boyatt and J. Sinclair, "Experiences of teaching a lightweight formal method," *Proceedings of Formal Methods in Computer Science Education*, vol. 16, 2008. 2.3.2

[63] J. Noble, D. J. Pearce, L. Groves, and Z. Istenes, "Introducing alloy in a software modelling course," *Formal Methods in Computer Science Education*, p. 81, 2008. 2.3.2

[64] M. Simonot, M. Homps, and P. Bonnot, "Teaching abstraction in mathematics and computer science-a computer-supported approach with alloy," in *International Conference on Computer Supported Education*, vol. 2, pp. 239–245, SciTePress, 2012. 2.3.2

[65] S. Tarkan and V. Sazawal, "Chief chefs of z to alloy: using a kitchen example to teach alloy with z," in *International Conference on Technical Formal Methods*, pp. 72–91, Springer, 2009. 2.3.2

[66] L. E. Brown, A. Feltz, and C. Wallace, "Lab exercises for a discrete structures course: Exploring logic and relational algebra with alloy," in *Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education*, ITiCSE 2018, (New York, NY, USA), p. 135–140, Association for Computing Machinery, 2018. 2.3.2

[67] N. Macedo, A. Cunha, J. Pereira, R. Carvalho, R. Silva, A. C. Paiva, M. S. Ramalho, and D. Silva, "Experiences on teaching alloy with an automated assessment platform," in *International Conference on Rigorous State-Based Methods*, pp. 61–77, Springer, 2020. 2.3.2

[68] N. Juristo and A. M. Moreno, *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st ed., 2010. 2.4.1, 3.4.1

[69] 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. 2.4.1, 3.4.1

[70] J. Siegmund, C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, "Measuring and modeling programming experience," *Empirical Software Engineering*, vol. 19, no. 5, pp. 1299–1334, 2014. 2.4.2, 4.4.4

[71] 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. 2.4.3, 3.4.2.1

[72] T. von der Malsburg, "Py-Span-Task – A Software for Testing Working Memory Span," June 2015. 2.4.3

[73] K. Wang, A. Sullivan, and S. Khurshid, "Arepair: a repair framework for alloy," in *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 103–106, IEEE, 2019. 2.4.3

[74] K. Wang, "Arepair: A repair framework for alloy - experiment repository." 2.4.3

[75] Alloy, "River Crossing Puzzle," 2022. 2.4.3

[76] J. Love, R. Selker, M. Marsman, T. Jamil, D. Dropmann, J. Verhagen, A. Ly, Q. F. Gronau, M. Šmíra, S. Epskamp, *et al.*, "Jasp: Graphical statistical software for common statistical designs," *Journal of Statistical Software*, vol. 88, pp. 1–17, 2019. 2.5.1, 4.5.2

[77] S. Pernsteiner, C. Loncaric, E. Torlak, Z. Tatlock, X. Wang, M. D. Ernst, and J. Jacky, "Investigating safety of a radiotherapy machine using system models with pluggable checkers," in *International Conference on Computer Aided Verification*, pp. 23–41, Springer, 2016. 2.7

[78] C. Sadowski, J. Van Gogh, C. Jaspan, E. Söderberg, and C. Winter, "Tricorder: Building a program analysis ecosystem," in *International Conference on Software Engineering*, pp. 598–608, IEEE, 2015. 3.1

[79] 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. 3.1

[80] 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*, pp. 1–8, ACM, 2007. 3.1

[81] A. Venet, "A practical approach to formal software verification by static analysis," *ACM SIGAda Ada Letters*, vol. 28, no. 1, pp. 92–95, 2008. 3.1

[82] N. Ayewah and W. Pugh, "The Google FindBugs fixit," in *International Symposium on Software Testing and Analysis*, pp. 241–252, ACM, 2010. 3.1

[83] 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. 3.1

[84] G. Brat and A. Venet, "Precise and scalable static program analysis of NASA flight software," in *Aerospace Conference*, pp. 1–10, IEEE, 2005. 3.1

[85] E. Denney and S. Trac, "A software safety certification tool for automatically generated guidance, navigation and control code," in *Aerospace Conference*, pp. 1–11, IEEE, 2008. 3.1

[86] X. Rival, "Abstract dependences for alarm diagnosis," in *Asian Symposium on Programming Languages and Systems*, pp. 347–363, Springer, 2005. 3.1

[87] 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*, pp. 462–473, ACM, 2015. 3.1

[88] I. Dillig, T. Dillig, and A. Aiken, "Automated error diagnosis using abductive inference," in *Conference on Programming Language Design and Implementation*, pp. 181–192, ACM, 2012. 3.1

[89] 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*, pp. 672–681, IEEE, 2013. 3.1

[90] 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*, pp. 470–481, IEEE, 2016. 3.1, 3.7.1.2

[91] 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. 3.1

[92] 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*, pp. 106–115, Sept 2013. 3.1

[93] 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*, pp. 176–185, IEEE, 2007. 3.1

[94] M. Christakis and C. Bird, "What developers want and need from program analysis: An empirical study," in *International Conference on Automated Software Engineering*, pp. 332–343, ACM, 2016. 3.1

[95] T. Muske, R. Talluri, and A. Serebrenik, "Repositioning of static analysis alarms," in *International Symposium on Software Testing and Analysis*, pp. 187–197, ACM, 2018. 3.1, 3.2, 3.7.1.1

[96] T. Muske, R. Talluri, and A. Serebrenik, "Reducing static analysis alarms based on non-impacting control dependencies," in *Asian Symposium on Programming Languages and Systems*, pp. 115–135, Springer, 2019. 3.1

[97] 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*, pp. 299–314, Springer, 2012. 3.1

[98] D. Zhang, D. Jin, Y. Gong, and H. Zhang, "Diagnosis-oriented alarm correlations," in *Asia-Pacific Software Engineering Conference*, pp. 172–179, IEEE, 2013. 3.1

[99] 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. 3.1

[100] T. Muske and A. Serebrenik, "Survey of approaches for postprocessing of static analysis alarms," *ACM Computing Survey*, vol. 55, Feb 2022. 3.2, 3.2

[101] Qualtrics, "Qualtrics xm - experience management software," May 2022. 3.4.1

[102] Amazon, "Amazon mechanical turk," May 2022. 3.4.1

[103] X. Ji, "Magpie experiments framework," June 2022. 3.4.2.1

[104] Magpie, "Mental rotation task," June 2022. 3.4.2.1

[105] N. Mansoor, "Operation span task," June 2022. 3.4.2.1

[106] 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*, pp. 537–548, IEEE, 2021. 3.4.2.2

[107] C. Wohlin, P. Runeson, M. Hst, M. C. Ohlsson, B. Regnell, and A. Wessln, *Experimentation in Software Engineering*. Springer Publishing Company, Incorporated, 2012. 3.4.2.3

[108] PrismJS, "Prismjs," May 2022. 3.4.5

[109] 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. 3.5.4

[110] S. Rugaber, "Program comprehension," *Encyclopedia of Computer Science and Technology*, vol. 35, no. 20, pp. 341–368, 1995. 4.2

[111] S. Letovsky, "Cognitive processes in program comprehension," *Journal of Systems and software*, vol. 7, no. 4, pp. 325–339, 1987. 4.2

[112] A. Von Mayrhauser and A. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, pp. 44–55, 1995. 4.2

[113] M.-A. Storey, "Theories, methods and tools in program comprehension: past, present and future," in *13th International Workshop on Program Comprehension (IWPC'05)*, pp. 181–191, 2005. 4.2

[114] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye movements in code reading: Relaxing the linear order," in *2015 IEEE 23rd International Conference on Program Comprehension*, pp. 255–265, IEEE, 2015. 4.2

[115] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, pp. 286–28610, 2018. 4.2

[116] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng, "A survey on the usage of eye-tracking in computer programming," *ACM Computing Surveys (CSUR)*, vol. 51, no. 1, p. 5, 2018. 4.2

[117] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A systematic literature review on the usage of eye-tracking in software engineering," *Information and Software Technology*, vol. 67, pp. 79–107, 2015. 4.2

[118] S. Stapleton, Y. Gambhir, A. LeClair, Z. Eberhart, W. Weimer, K. Leach, and Y. Huang, "A human study of comprehension and code summarization," in *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, (New York, NY, USA), p. 2–13, Association for Computing Machinery, 2020. 4.2

[119] R. Bednarik, C. Schulte, L. Budde, B. Heinemann, and H. Vrzakova, "Eye-movement modeling examples in source code comprehension: A classroom study," in *Proceedings of the 18th Koli Calling International Conference on Computing Education Research*, Koli Calling '18, (New York, NY, USA), Association for Computing Machinery, 2018. 4.2

[120] A. A. Shargabi, S. A. Aljunid, M. Annamalai, and A. M. Zin, "Performing tasks can improve program comprehension mental model of novice developers: An empirical approach," in *Proceedings of the 28th International Conference on Program Comprehension*, ICPC '20, (New York, NY, USA), p. 263–273, Association for Computing Machinery, 2020. 4.2

[121] T. D. LaToza, D. Garlan, J. D. Herbsleb, and B. A. Myers, "Program comprehension as fact finding," in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pp. 361–370, 2007. 4.2

[122] C. L. Corritore and S. Wiedenbeck, "What do novices learn during program comprehension?," *International Journal of Human-Computer Interaction*, vol. 3, no. 2, pp. 199–222, 1991. 4.2

[123] J.-M. Burkhardt, F. Détienne, and S. Wiedenbeck, "Object-oriented program comprehension: Effect of expertise, task and phase," *Empirical Software Engineering*, vol. 7, no. 2, pp. 115–156, 2002. 4.2

[124] A. D. Baddeley, "Working memory," *Philosophical Transactions of the Royal Society of London. B, Biological Sciences*, vol. 302, no. 1110, pp. 311–324, 1983. 4.2

[125] C. N. Arrington, P. A. Kulesz, D. J. Francis, J. M. Fletcher, and M. A. Barnes, "The contribution of attentional control and working memory to reading comprehension and decoding," *Scientific Studies of Reading*, vol. 18, no. 5, pp. 325–346, 2014. 4.2

[126] H. E. Gardner, *Frames of mind: The theory of multiple intelligences*. Hachette Uk, 2011. 4.2

[127] Eclipse, "Eclipse IDE for C/C++ Developers | Eclipse Packages." 4.4.1

[128] B. Sharif, C. Peterson, D. Guarnera, C. Bryant, Z. Buchanan, V. Zyrianov, and J. Maletic, "Practical eye tracking with itrace," in *2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)*, pp. 41–42, IEEE, 2019. 4.4.1, 4.4.3

[129] B. Sharif and J. I. Maletic, "itrace: Overcoming the limitations of short code examples in eye tracking experiments.," in *ICSME*, p. 647, 2016. 4.4.1, 4.4.3

[130] B. Sharif, T. Shaffer, J. Wise, and J. I. Maletic, "Tracking developers' eyes in the ide," *IEEE Software*, vol. 33, no. 3, pp. 105–108, 2016. 4.4.1, 4.4.3

[131] V. Zyrianov, C. S. Peterson, D. T. Guarnera, J. Behler, B. Sharif, J. I. Maletic, *et al.*, "Deja vu: semantics-aware recording and replay of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks—methodology and analyses," *Empirical software engineering*, vol. 27, no. 7, pp. 1–39, 2022. 4.4.3

[132] N. Mansoor, "Participate in SERESL Lab eye tracking study | Announce | University of Nebraska-Lincoln." 4.4.4

[133] Z. Sharafi, T. Shaffer, B. Sharif, and Y.-G. Guéhéneuc, "Eye-tracking metrics in software engineering," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*, pp. 96–103, IEEE, 2015. 4.4.5

[134] K. Rayner, "The 35th sir frederick bartlett lecture: Eye movements and attention in reading, scene perception, and visual search," *Quarterly journal of experimental psychology*, vol. 62, no. 8, pp. 1457–1506, 2009. 4.4.5

[135] M. L. Collard, M. J. Decker, and J. I. Maletic, "srcml: An infrastructure for the exploration, analysis, and manipulation of source code: A tool demonstration," in *2013 IEEE International Conference on Software Maintenance*, pp. 516–519, IEEE, 2013. 4.5.1

[136] C.-K. Yang and C. Wacharamanotham, "Alpscarf: Augmenting scarf plots for exploring temporal gaze patterns," in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI EA '18, (New York, NY, USA), p. 1–6, Association for Computing Machinery, 2018. 4.5.3

[137] G. Zheng, T. Nguyen, S. Gutiérrez Brida, G. Regis, M. F. Frias, N. Aguirre, and H. Bagheri, "Flack: Counterexample-guided fault localization for alloy models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 637–648, 2021. 5.1.3

[138] T. R. Tulili, A. Capiluppi, and A. Rastogi, "Burnout in software engineering: A systematic mapping study," *Information and Software Technology*, p. 107116, 2022. 5.2.2