Sander Bjerklund Lindberg

Exposing novice programmers to an expert's eye-gaze

Investigating the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novice programmers how to comprehend and debug code

Master's thesis in Informatics
Supervisor: Kshitij Sharma
June 2022

Sander Bjerklund Lindberg

Exposing novice programmers to an expert's eye-gaze

Investigating the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novice programmers how to comprehend and debug code

Master’s thesis in Informatics
Supervisor: Kshitij Sharma
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Computer Science

NTNU
Norwegian University of
Science and Technology

Abstract

As programming, code debugging, and code comprehension are cognitive processes, it may be difficult for an individual to discern their exact thought process in order to teach others. Through the usage of eye-trackers, researchers have found a way to visualize these cognitive processes. Furthermore, they have identified differences in how experts and novices solve problems. This thesis investigates the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novices how to comprehend and debug code.

This has been done through the execution of a study with 32 novice programmers currently studying computer science. At first, a system for presenting code snippets and eye-gaze was created in React.js and Node.js. Then, an expert's eye-gaze was recorded while they completed a set of code problems. Later, the 32 novices completed the same problems, with half of the participants able to see the expert's eye-gaze and code snippets side-by-side while solving the problems. In addition to being split into a control group and a help group, participants were split into groups of expertise and performance to investigate the effect of the expert's eye-gaze.

The analysis results showed no significant differences between the group exposed to the expert's eye-gaze and the group not exposed regarding scores on the problems. Moreover, the results showed no significant difference in number of fixations on lines with bugs, nor the fixation duration on them, among the participants. However, the results did show a significant difference between participants concerning task time and fixation distance to meaningful areas in the code.

Sammendrag

Siden programmering, kodefeilsøking og kodeforståelse er kognitive prosesser, kan det være vanskelig for en person å skjelne sin nøyaktige tankeprosess for å lære bort hvordan de selv løser oppgaver til andre. Gjennom bruk av eye-trackere har forskere funnet en måte å visualisere disse kognitive prosessene. Videre har de identifisert forskjeller i hvordan eksperter og nybegynnere løser problemer. Denne masteroppgaven undersøker effekten av å visualisere en ekspertprogrammerers kognitive kodeforståelse og feilsøkingsprosess ved bruk av øyesporing, som grunnlag for å lære nybegynnere hvordan de skal forstå og feilsøke kode.

Dette har blitt gjort gjennom utførelse av en studie med 32 nybegynnere i programmering, som for tiden studerer innenfor datateknologi. Til å begynne med ble et system for presentasjon av kodebiter og blikk laget i React.js og Node.js. Deretter ble en eksperts blikk sporet mens de fullførte et sett med kodeproblemer. Senere fullførte de 32 nybegynnerene de samme kodeproblemene, hvor halvparten av deltakerne hadde muligheten til å se ekspertens blikk på skjermen og kodesnutter side ved side mens de løste problemene. I tillegg til å bli delt inn i en kontrollgruppe og en hjelpegruppe, ble deltakerne delt inn i ekspertise- og prestasjonsgrupper for å utforske effekten av ekspertens blikk.

Resultatene av analysen viste ingen signifikante forskjeller mellom gruppen som ble utsatt for ekspertens blikk og gruppen som ikke ble eksponert, med hensyn til skår på problemene. Dessuten viste resultatene ingen signifikant forskjell mellom deltakerene, med hensyn til antall fikseringer på linjer med feil, eller fikseringsvarigheten på dem. Resultatene viste imidlertid en signifikant forskjell mellom deltakerne med hensyn til oppgavetid og fikseringsavstand til meningsfulle områder i koden.

Preface

Ever since primary school, I knew I wanted an education in Computer Science. The memory of my father setting up and installing my first ever computer as I lay in bed watching him, refusing to go to sleep, is still strong.

I've come a long way since then. After finishing High School, where my devious little self had persuaded enough of my classmates to choose IT as an elective course so that the course would have enough pupils to be lectured, it was finally time to get my Computer Science education. The choice was easy - I had to apply to the Norwegian University of Science and Technology, one of Norway's largest universities. I was accepted and finished my BSc in informatics within three years. Still, I did not feel my education was complete and applied for a two-year MSc program in Informatics, specializing in artificial intelligence.

Throughout my Bachelor's and Master's degrees, I have worked as a Teaching Assistant in multiple courses. This has given me an insight into the difference between students and made me aware that I enjoy teaching. The original plan for my Master's thesis was to predict student performance using eye-tracking. However, after reading literature, conversing with my supervisor, and my experience as a TA, I quickly realized that I wanted to explore teaching methods.

This thesis concludes my Master of Science in Informatics at the Department of Computer Science at the Norwegian University of Science and Technology in Trondheim. It focuses on visualizing an expert programmer's eye-gaze to novices in programming to explore a new method of teaching novices code comprehension and debugging.

The road to get here has been long. It's been fun, frustrating, and knowledgeable. Tiresome, motivating, and exciting. I want to express my thanks to NTNU for giving me the Computer Science education I've always dreamed of and for preparing me for the new chapter in my life that's now coming. I would also like to thank my supervisor Kshitij Sharma for his knowledge, feedback, and motivation throughout this thesis. Furthermore, I want to express my gratitude to my family and friends for supporting me all these years, and lastly, my partner for putting up with my constant nagging.

Sander Bjerklund Lindberg
Trondheim, June 12, 2022

Table of Contents

List of Figures

List of Tables

Acronyms

AOI	Area of interest.
CS	Computer Science.
JSON	JavaScript Object Notation.
SQL	Structured Query Language.
TA	Teaching assistant.

Chapter 1

Introduction

1.1 Background and motivation

Since the dawn of time, the challenge of problem solving, especially the optimization of this process, has puzzled humankind. The ancient Greeks attempted to automate this process by creating algorithms, such as the Pythagorean theorem, to more easily solve repetitive problems. However, since problems vary greatly, from repetitive mathematical equations to programming fully conscious AI, there is still a need to find more efficient ways to teach problem solving.

In Merriam-Webster, problem solving is defined as ”The process or act of finding a solution to a problem” [1]. This process is highly individual and varies greatly. Since there are often many ways to solve a given problem, conveying the different individual processes of finding a solution can be challenging. This is especially apparent in a teaching context, where students might have problems understanding their teacher's thought process when solving, for instance, mathematical equations. Through the usage of eye-trackers, Yoon and Narayanan [2] managed to capture this cognitive process. They captured people's eye movements during a problem solving session and observed different problem solving processes.

Programming is about finding a solution to a problem, and it most certainly falls under the definition of problem solving. When teaching programming, one seeks to teach specific approaches to solving problems. Like the Pythagorean formula in mathematics helps identify the lengths of the sides of triangles, programmers are thought if-statements to solve logical problems. Another form of problem solving in programming is code comprehension and debugging. As Yoon and Narayanan [2] found, there are several different problem solving processes, which also apply to code comprehension and debugging.

Through the usage of eye-trackers, it has been found that expert and novice programmers have different processes when it comes to code comprehension [3, 4, 5] and debugging [6, 7]. However, as these processes are cognitive and cannot be seen, it is hard to teach them to novice programmers, especially when they, more often than not, have different processes of comprehension and debugging than the expert or teacher.

This raises the question of whether one can visualize an expert's problem solving process through their eye-gaze to better teach novice programmers their approach. This thesis explores previous work utilizing eye-trackers and describes the design and execution of a study investigating what effect exposing novices to an expert's eye-gaze has on their code comprehension and debugging abilities.

1.2 Research questions and objectives

The main goal of this thesis is to investigate the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novice programmers how to comprehend and debug code. In other words, this thesis will visualize an expert programmer's eye-gaze from comprehension and debugging sessions to novice programmers to lay the groundwork for further research on implementing eye-tracking and visualization of problem solving processes in computing education.

In order to achieve this goal, the following research questions will be answered:

RQ1 How is eye-tracking used in programming?

RQ2 What are the differences between experts and novices when programming?

RQ3 What effect does exposing novice programmers to an expert programmer's eye-gaze have on their comprehension and debugging abilities?

RQ3.1 What is the effect of exposing novice programmers to an expert programmer's eye-gaze during debugging and code comprehension?

RQ3.2 What are the differences between high and low expertise novice programmers when exposed to an expert programmer's eye-gaze?

RQ3.3 What are the differences between high and low performance novice programmers when exposed to an expert programmer's eye-gaze?

1.3 Research method

In order to answer the research questions in section 1.2, the first step was to conduct a literature study to get an overview of the existing literature and research in the field of eye-tracking and programming. The literature study was conducted in the fall of 2021 as a preparatory project for this thesis and is explained in more detail in section 2.3.

After the literature study, it became apparent that the field lacked research regarding helping novices learn programming and how one can use eye-tracking as an aid in learning situations. In addition, the literature study showed significant differences between novices and experts when comprehending and debugging source code. This spawned the interest in the design and execution of a study investigating what effect exposing novice programmers to an expert's eye-gaze has on their code comprehension and debugging abilities, to investigate if eye-tracking could be incorporated into university teaching situations.

1.4 Thesis structure

This thesis presents a short background information on eye-tracking terminology in section 2.1, followed by related work in section 2.3. Chapter 3 presents the design and execution of a study investigating the effects of exposing novice programmers to an expert programmer's eye-gaze, with section 3.1 presenting the design and architecture of the system used in the study, section 3.2 describing the design and execution of the study and section 3.3 giving a detailed presentation of the analysis. Chapter 4 details the analysis results and is followed by a discussion and possible aspects to investigate in future work in chapter 5. Finally, a conclusion is drawn in chapter 6.

Chapter 2

Background and related work

This chapter is imported in it's entirety from my preparatory project "How eye tracking can be used in problem solving" [8] *with modifications and additions.*

2.1 Eye-tracking terminology and technology

Eye-tracking is recording and tracking a person's eye movements as they view, e.g. objects, lines of text, or other visual stimuli on a screen. As eye movements carry visual attention of stimuli to parts of the brain, they are essential to cognitive processes in human vision [9]. Eye-*trackers* are designed to monitor and capture such movements so that it is possible to study how a person looks at objects on a screen. Eye-trackers can come in many different forms. Some are screen-based eye-trackers mounted to a computer screen. Others are glasses for mobility, an integrated or embedded system such as VR glasses, or head stabilized eye-trackers where the person's head is stabilized by resting the chin and forehead in a device so that the head will not move. Eye movement data is often studied with a focus on Area of interest (AOI). An AOI is often defined as a subsection of the viewed screen. It can, for instance, be specific parts of a viewed image, specific paragraphs of a written text, or source code elements such as method signatures or line numbers. Not all AOIs are necessarily relevant to the task being performed. For instance, if the task is an object-oriented debugging task, multiple classes could be

AOIs, but only one class could be relevant as it contained the bug.

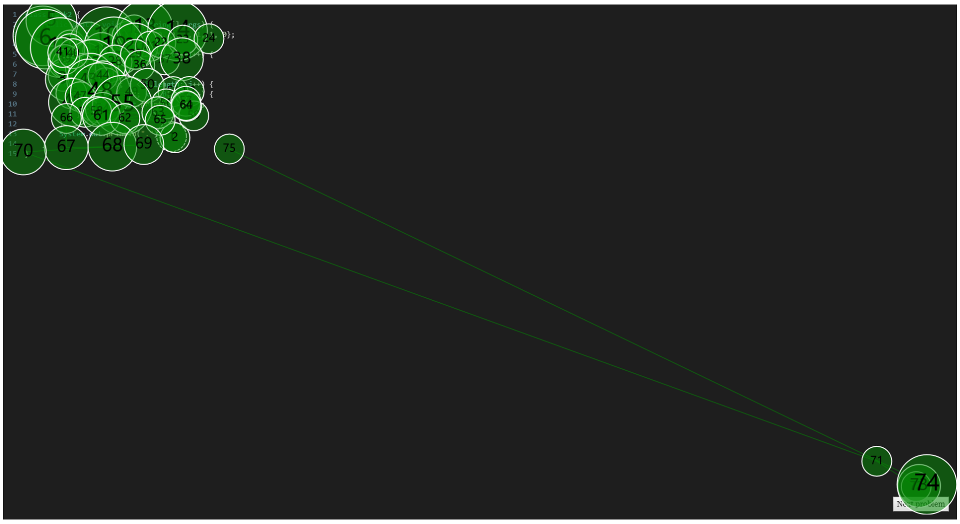


Figure 2.1: Example fixations and scanpath. The green circles are fixations, and the lines between them are saccades. The saccades and fixations make up a scanpath.

Figure 2.1 gives an example of fixations, saccades, and scanpath that will be presented in this paragraph. Eye-tracking data are divided into different types of metrics; fixations, saccades, scanpaths, and the aforementioned AOIs [10]. A fixation is a period of time when the eyes are focused on the same area, from a few hundred milliseconds to multiple seconds. A longer fixation may thus indicate that the area looked at is more interesting or complex. Fixation as a metric can be used for multiple purposes, and investigating their length or how many fixations occur on an AOI may provide interesting insights. The next metric, saccades, are rapid eye movements from one fixation to another. A saccade is simply rapid eye movements where no visual information is processed. An eye-tracker that samples at 50Hz or greater are needed to detect saccades. In programming, saccades can occur when the eyes move between connected lines. Scanpaths are a combination of fixations and saccades, showing the entire eye-gaze path.

2.2 Zone of Proximal Development

The zone of proximal development is a concept introduced by Lev Vygotsky. In Mind in society: Development of higher psychological processes, he defined it as:

"[...] the distance between the actual developmental level as determined by independent problem solving and the level of potential development as determined through problem solving under adult guidance or in collaboration with more capable peers."[11, p. 86]

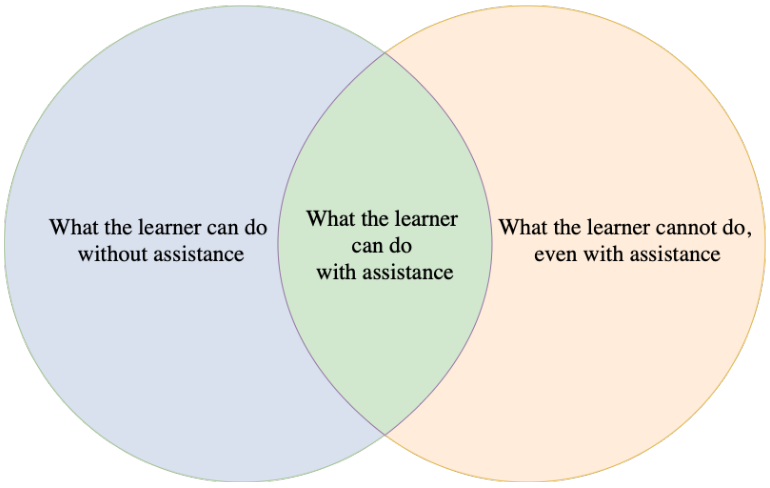

What the learner can do without assistance
What the learner can do with assistance
What the learner cannot do, even with assistance

Figure 2.2: Vygotsky's Zone of Proximal Development can be thought of as the intersection between what a learner can do by themselves and what the learner cannot do, even with assistance. Modified from Caspersen et al. [12].

In other words, the zone of proximal development is a state children, or students, can be in where they might not be able to solve a problem independently but will be able to through guidance from an adult or more advanced peers. For instance, in a computer science class, a student may have learned how to write a single for-loop but struggle to understand nested loops. A Teaching assistant (TA) can show an example of a nested loop and provide hints when the student gets stuck. Upon being given some examples, the student may then be able to solve the problem. The nested loops example above is also an example of a *problem* that might be in a *student's* zone of proximal development.

Through Vygotsky's work, Doolittle [13] argued that a child develops cognitively by first being exposed to tasks or situations in the upper end of the zone of proximal development. These tasks would at first require assistance in order for the child to be able to complete them but would eventually require less and less assistance. This thesis describes the execution and findings of a study that aims to take advantage of Vygotsky's zone of proximal development and Doolittle [13]'s argument of exposure to tasks or situations in the upper end of the zone of proximal development.

2.3 Related work

This literature study was carried out as a basis and preparatory project [8] for this thesis in the fall of 2021. It seeks to answer the following research questions:

RQ1 How is eye-tracking used in programming?

RQ2 What are the differences between experts and novices when programming?

Even though this literature study was not a *structured literature review*, it followed a similar structure as the one proposed by Kitchenham [14]. First, several published papers were identified using search strings connected to eye-tracking, programming, and learning. The titles of the resulting articles were then, subjectively, rated relevant or irrelevant. Then, related articles were found. Next, the abstracts of the relevant titles were read, and irrelevant pieces were discarded. Finally, the reference list of the selected papers was analyzed, and relevant articles were extracted from them.

An overview of the eye-trackers used by the explored studies can be seen in Table 2.1 (if applicable).

Table 2.1: Overview of different eye-trackers used by the related works' studies.

Eye-tracker used	**Study**
SMI Eye Link	Yoon and Narayanan [2]
SMI RED-m 120 Hz	Busjahn et al. [15]
Tobii TX300	Najar et al. [4]
Tobii X60	Kevic et al. [16]
Smart Eye Aurora	Jessup et al. [5]
ISCAN RK-726PCI	Stein and Brennan [7]
Tobii 1750 50hz	van Gog et al. [17]

2.3.1 Eye-tracking in problem solving

Merriam-Webster defines problem solving as ”The process or act of finding a solution to a problem” [1]. This process of finding a solution to a problem varies with different people. For example, in maths, when adding $59 + 41$, one individual might instantly think to find the answer by adding 50 to 40 and then adding nine to one. Meanwhile, another may add 59 to one and then add 40. These processes can be hard to observe without verbal explanation, and it is individual which process one prefers. Therefore, the process of learning and thus teaching how to solve problems is highly subjective. Yoon and Narayanan [2] did an eye-tracking study where they wanted to look for evidence and effect of a mental imagery problem solving process. They describes mental imagery as ”[...] the phenomenon in which someone imagines an object or a visual scene in his or her ”mind’s eye” in order to retrieve information from that mental image or to transform it so as to generate needed information.” Their study contained 90 engineering students who were given two problems to solve. The first problem contained an image and a question related to this image. After answering the first question, the system displayed the second problem, which was also related to the image. However, in this question, the subjects could not see the image. The question was placed at the same place as the first question, leaving a blank area where the image used to be.

After the experiment, the researchers found that some subjects gazed at the blank section of the second screen where the image used to be, as though there was still an image there. These subjects were categorized as imagery strategy behavior, whereas the rest were classified as non-imagery. Their results show no significant difference between the two groups regarding accuracy on the two problems. However, they did find that the imagery group had a higher mean fixation duration on the critical component. Furthermore, when comparing the imagery subjects that answered problem two correctly and incorrectly, they found that the subjects with a correct answer had significantly higher coverage and more time fixating on the critical component. These results suggest that higher accuracy can be improved by paying attention to critical components.

Yoon and Narayanan [2] is thus an example of how we can observe different problem solving strategies by using eye-trackers. It is also an example of how attention to critical parts of a problem can be essential to solving the given problem.

2.3.2 Eye-tracking in programming

According to Sharafi et al. [9] and their literature review on the topic, 36 relevant papers on the subject of eye-tracking in software engineering was published from 1990 to 2015, with 86.2% of these published after 2006. This suggests an increasing interest and acceptance in using eye-trackers in programming.

Busjahn et al. [15] are some of the researchers who investigated eye-tracking and software engineering. I their paper ”Eye tracking in Computing Education,” Busjahn et al. [15] introduce how eye-tracking can be used as an instrument for computer science education research. They analyzed data from a study with two professional software developers reading and understanding short Java programs. The subjects were informed that the code did not contain any bugs and that their task was comprehending it. Subject 1 was told they would get questions regarding the return value of one method, whereas subject two was told to expect a multiple-choice test about the whole algorithm. After analyzing the eye-gaze of the two subjects, the authors identified 11 new eye-gaze patterns in source code reading that can be used when researching software engineering in combination with eye-tracking. In addition, they identified 14 different strategies which programmers use when comprehending code. Their research opens up new perspectives for teaching code reading and code comprehension. Eye-gaze from students can be used to identify differences in strategies or make a novice aware of how they go about solving a task by, for instance, having a novice record themselves when reading code and later reviewing their eye-gaze with or without a teacher present.

2.3.3 Determining programming expertise

Code comprehension is often used as a measurement when conducting studies containing software engineers. Since this is a cognitive process, observing it from an outside perspective is difficult. Instead, one may have the subjects verbally explain, observe their behavior, or answer a questionnaire afterward. In order to do so, independent variables have to be controlled. This could, for instance, be done by including programming experience. A more experienced programmer will likely have a deeper understanding of the code than an inexperienced programmer. Suppose one were to conduct a programming study containing a surplus of experienced programmers. In that case, this could lead to biased results as the more experienced programmers have a better basis for solving programming tasks. It is therefore important to have a way to measure programming expertise.

Siegmund et al. [18] states that there is no agreed way to measure programming expertise and that a common understanding of programming expertise can increase the validity of experiments. Therefore, to get an overview of how researchers measure programming experience, they conducted a literature review of highly ranked conference - and journal papers between 2001 and 2010. In addition, they conducted an experiment in which subjects solved programming tasks and completed a questionnaire with questions found in literature related to programming experience. As a result, they were left with a questionnaire that contained questions that should be able to measure programming experience and a reusable experimental design to evaluate the questionnaire.

2.3.4 Difference between novices and experts

Crosby and Stelovsky [3] were pioneers in using eye-trackers in programming research [15]. In 1990, they wanted to research how programmers read algorithms. This was done by conducting a study with students of different expertise from a computer science program at the University of Hawaii. Each participant was given a textual slide of pascal code and was told to read and understand the code. After indicating they were done reading, the participants were asked to correct the code if necessary. Their results show that low experience programmers tend to spend significantly more time reading comments and comparisons than highly experienced programmers. In contrast, the high experience programmers tend to spend significantly more time reading complex statements. In addition, they found that the highly experienced subject recognized and spent more time concentrating on meaningful areas of the code.

As demonstrated by Crosby and Stelovsky [3], eye-trackers can provide an insight into the difference between novices and experts. Another example is given by Najar et al. [4], who utilized eye-tracking to improve learning by examples. They conducted a study on 42 students currently studying Structured Query Language (SQL). The students were presented with a worked example in the context of SQL-tutor, a system used for learning and teaching SQL. The software window was divided into four sections; an example SQL code, an explanation of the code, the database schema, and a multiple-choice questionnaire to the example. For each example, the students were presented with the same window layout. After each multiple-choice submission, the system presented the correct answer.

By using previous test scores, Najar et al. [4] could divide the students into groups of novices and advanced students. To see how the students studied the examples, AOIs were defined for the systems interface, namely the four sections of the software mentioned above. Through the students' eye movements, the researchers found several different transition patterns between the four AOIs. What is interesting is the difference in these patterns between the novices and advanced students. The most interesting finding was that the advanced students looked significantly more at the database schema than the novices (90% against only 25%). Furthermore, the novices did not use the pattern Najar et al. [4] called ED, which was a gaze from the example to the database schema.

Sharma et al. [19] did a study on high performance and low performance learners. They were specifically interested in how facial expressions could explain learning performance in different learning situations, such as collaborative programming.

This was accomplished by extracting emotions such as boredom, delight, frustration, and confusion from 13-16-year-old children's facial expressions during collaborative game creation with Scratch. The study was placed at NTNU's premises in Trondheim, in an informal environment. The main goal was to introduce computer science and programming to school children who previously had no coding experience. They measured both emotions and the transition between emotions, for example, the transition from frustration to delight. Every 45 minutes, they collected four game versions from each team. This laid the basis for their separation into low and high performing teams. If a team had higher than the median points (analyzed by DrScratch[1]) in two of the four phases, they were labeled as a high performing team and low performing if not.

Their findings show a difference in emotions between the high and low performing teams. The high performance teams displayed a significantly higher amount of confusion and frustration than the low performance teams, and the low performance more often displayed signs of boredom than the high performance. Both teams showed an equal amount of delight and neutral. The teams did also show a difference in transitioning between emotions. High performance teams had a much higher probability of transitioning from boredom to neutral than low performance teams, which had a higher probability of transitioning to frustration. The low performance teams also showed a much higher probability of transitioning from confusion to boredom than the high performance teams, with probabilities of ca $0.5 - 0.6$ and ca $0.1 - 0.15$, respectively. Most interesting is that for all emotions, the low performing teams had the highest probability of transitioning to boredom, whereas the high performance teams had a mix between neutral and frustration.

Aljehane et al. [6] studied the difference in source code reading behaviors between experts and novices in Java by investigating eye movements. They analyzed a dataset already collected by Kevic et al. [16] in 2015 through the Eclipse plugin iTrace [20]. The subjects in the study were 12 industry-working programmers, the experts, and ten Computer Science (CS) students, the novices. They were given three different debugging tasks with varying difficulty ranging from missing commas and multiple classes to fixing a failure to launch the Acrobat on Win98. The eye-tracker used can be seen in table 2.1. Aljehane et al. [6] used the data related to a task that only required the participants to read one class. In addition, they only used the eye-tracking data from eighteen of the twenty-two participants, as four of them did not look at the class where the bug was found. Their results showed that novices read more source code elements than experts overall.

[1]`http://www.drscratch.org/`

Jessup et al. [5] also studied the difference between novices and experts during programming. Rather than looking at reading behaviors and debugging, as Aljehane et al. [6] did, they focused on code comprehension and the difference in fixation counts between experts and novices. They hypothesized that experts would have a lower fixation count than the novices and more frequently describe code functions. A total of 36 participants, 22 novices and 14 experts, were included in the study. The participants were given two tasks of code comprehension in a within-subjects manner. Even though they hypothesized that experts would have a lower fixation count than the novices, their findings showed this was not the case. They found significant differences between the experts and novices considering fixation count. However, contrary to their hypothesis, experts had the highest fixation count. Furthermore, their second hypothesis that the experts accurately describe code more frequently did not hold. The results showed no significant difference between the two groups.

Turner et al. [21] had another approach to investigate the difference between expertise groups. They wanted to examine the differences in reading behaviours across programming languages, specifically Python and C++. Their study consisted of 38 participants, both novices and experts. The participants were split into two groups. One group read Python code, and the other read C++ code. Once divided, they were told to comprehend code snippets and verbally explain what the code did. In addition, they were told to detect bugs and present how they would fix them. Their findings show that, within the Python group, the experts spent significantly more time than the novices. In contrast, the opposite was true for the C++ group. Furthermore, they found that within the Python group, the novices had a lower fixation rate on lines with bugs than the experts and that the fixation duration on the lines with bugs was, for the most part, the same for the two expertise groups.

All papers have found interesting differences between low and high performance students and expert's and novices. Najar et al. [4] found that low performance students did not use the database schema at all, and Sharma et al. [19] found that low performance students experienced much more boredom than high performance students. It may be possible that the low performance students in Najar et al. [4] experienced boredom and therefore chose not to look at the database schema to get through the questions more rapidly. Questions can be raised as to why low performance students experience more boredom. Is it because they do not possess the necessary basic knowledge surrounding the subject, or are they not interested in it? This can lead to interesting research on how teachers can use eye-tracking and facial expressions to facilitate their teaching better or set up software layouts. High performance students may then play a role in where to place certain items on a screen, I.e., the database schema, so that it is more visible and students are forced to read it. Aljehane et al. [6] showed that expert programmers look less at source code elements than novices. Based on what expert programmers are looking at, one could teach novice programmers which aspects or code snippets to spend more time on. In combination with this, Jessup et al. [5] showed that experts have higher fixation counts than novices. This again leads to interesting questions such as whether experts are better at singling out relevant code snippets and fixating on these and if this can help novices become better programmers.

2.3.5 Using other people's eye-gaze

As explained in the previous subsections, eye-trackers can provide different insights in, e.g., how novices differ from experts or as an instrument for computer science education research. However, Stein and Brennan [7] takes the eye-trackers one step further and exposes ”regular people” and not only researchers to the eye-tracking data. They conducted a study consisting of two phases. The study had ten participants who were all recent computer science graduates (within one year) from different schools, working professionally as software engineers. In the study's first phase, the participants wore head-mounted eye-trackers, which tracked their eye-gaze while debugging and comprehending three different Java programs. In phase two, six other participants were told that they would be watching videos of other programmers finding some (but not all) the bugs themselves would have to discover. The participants in phase two were randomly assigned to one of three eye-gaze recordings, which were viewed *before* they started identifying the bugs. The participants in phase two were also split into two groups. One group watched the eye-gaze video assigned to them before the first four bugs, and the second group before the last four bugs.

The main goal of Stein and Brennan's study was to test whether seeing another person's eye-gaze while debugging can be helpful to another person doing the same task. Their results show that the participants in phase two who saw the eye-gaze did, in fact, find bugs faster than the ones who did not. Furthermore, the eye-gaze did provide an average advantage of 62 seconds for finding bugs. There were, however, some varying results. For example, one participant spent more time finding one of the bugs after seeing the eye-gaze. He explained that he had trouble remembering where the eye-gaze ended.

van Gog et al. [17] also exposed people to another person's eye-gaze. They differ from Stein and Brennan [7] in that the task was not a programming task but a problem-solving task. They wanted to answer whether attention guidance by showing students a model's eye movements can enhance their learning. The study contained 77 participants from the University of Tübingen that had no prior knowledge of the task. They did a 2x2 factorial design, with the example type (product-oriented (no verbal help from the model) vs. process-oriented (verbal help from the model)) and attention guidance (yes vs. no) as factors. The model in this study was an expert in performing the task. The problem to be solved is known as Frog Leap[2], where the main goal is to switch the frogs' sides. The problem has only one solution, which can be applied by starting at either frog-side. This is what the participants did. First, they underwent a learning phase, where they studied their respective examples with or without attention guidance. Afterward, they tried to solve the problem themselves two times. One group attempted to solve the problem starting from the left and another from the right, as in the examples. Even though the authors hypothesized that attention guidance would be helpful for learning, they found that showing the model's eye movement hampered learning. They also found that relatively more students with product-oriented examples and attention guidance solved the second problem, in contrast to the first. They argue that, even though this finding should be interpreted carefully, this might suggest that the effect of attention guidance only becomes apparent on transfer tasks.

[2]`http://www.jwstelly.org/LeapFrog.html`

Chapter 3

Methodology

As explained in subsection 2.3.5 above, presenting people with another person's eye-gaze can improve performance and debugging time. Stein and Brennan [7] however, found that the advantage of seeing an eye-gaze before doing a task is limited to remembering the eye-gaze. Furthermore, van Gog et al. [17] argued that the effect of attention guidance might only be apparent on transfer tasks. In addition, Najar et al. [4] and Jessup et al. [5] both found differences between experts and novices when debugging and comprehending code.

This section contains the design and execution of a study that overcomes the limitation of remembering the eye-gaze by presenting it and the task side-by-side. Furthermore, the study utilizes Stein and Brennan [7] future works section to see whether viewing an expert's eye-gaze can provide interpretable cues to novices. Moreover, it investigates if this can decrease the difference in transition patterns found between experts and novices by Najar et al. [4]. The study also examines whether or not exposing a novice to an expert's eye-gaze can help the novice identify bugs with reduced time spent reading source code elements compared to the controls, seeing as Jessup et al. [5] found that novices spent more time looking at source code compared to experts.

This chapter presents the architecture and design of a system created to present tasks and visualize an expert's eye-gaze in section 3.1. Section 3.2 describes the study's planning, design, and execution, and section 3.3 presents detailed methods of analysis conducted on the collected data.

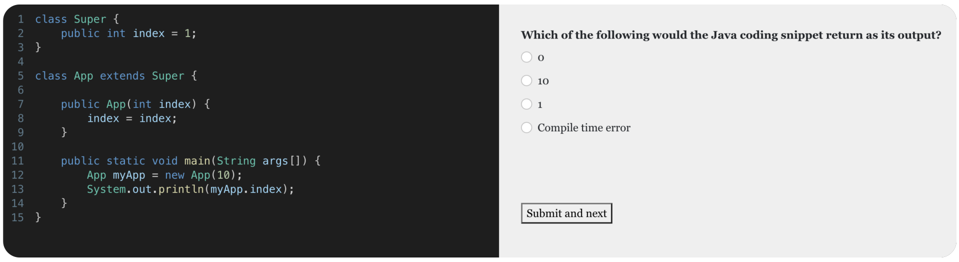
3.1 System design

To conduct the study described in section 3.2, a system was made to execute the pretest and tasks and visualize the expert's eye-gaze.

The system was a self-developed React[1] application [22], with a Node[2] server [23] running backend. The application was running at localhost during the study, and participants' answers and events were logged and stored locally on a pc at NTNU premises. This subsection contains a description of the system used in the study, including mockups and the system's flow.

3.1.1 System mockups

As the study included both experts and novices, two versions of the system were made. One version had a pretest with ten questions and code snippets and one without. In addition, both versions incorporated six larger code snippets with associated debugging and comprehension questions.



```java
class Super {
    public int index = 1;
}

class App extends Super {

    public App(int index) {
        index = index;
    }

    public static void main(String args[]) {
        App myApp = new App(10);
        System.out.println(myApp.index);
    }
}
```

Which of the following would the Java coding snippet return as its output?

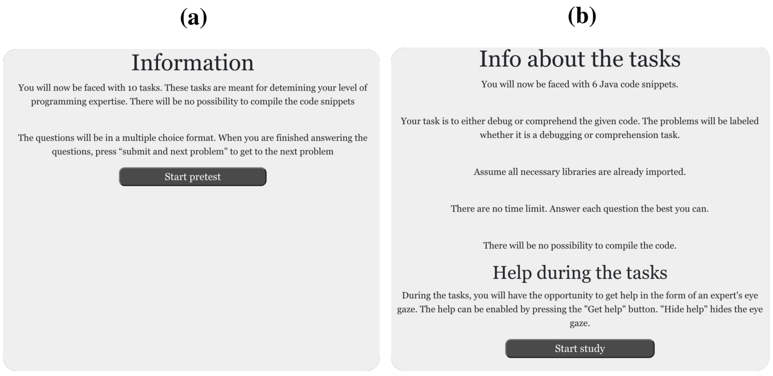
- 0
- 10
- 1
- Compile time error

Figure 3.1: Pretest example. The screen is divided in two, with the code snippet on the left and a multiple choice question to the code on the right. The image is cropped vertically.

As mentioned, the participants answered a pretest that maps their debugging and comprehension knowledge to determine further if they were a novice or not. The pretest had the same layout as the actual tasks so that the participants would get familiar with the look and feel of the system. See Figure 3.1 for an example of a pretest task-page.

The system included two information pages. One for the pretest and one for the study. The information given can be seen in Figure 3.2 below.

[1]`https://reactjs.com`. Version 17.0.2 was used in building the system in this thesis.
[2]`https://nodejs.org/en/`. Version 16.9.1 was used in building the system in this thesis.



Information

You will now be faced with 10 tasks. These tasks are meant for detemining your level of programming expertise. There will be no possibility to compile the code snippets

The questions will be in a multiple choice format. When you are finished answering the questions, press "submit and next problem" to get to the next problem

Info about the tasks

You will now be faced with 6 Java code snippets.

Your task is to either debug or comprehend the given code. The problems will be labeled whether it is a debugging or comprehension task.

Assume all necessary libraries are already imported.

There are no time limit. Answer each question the best you can.

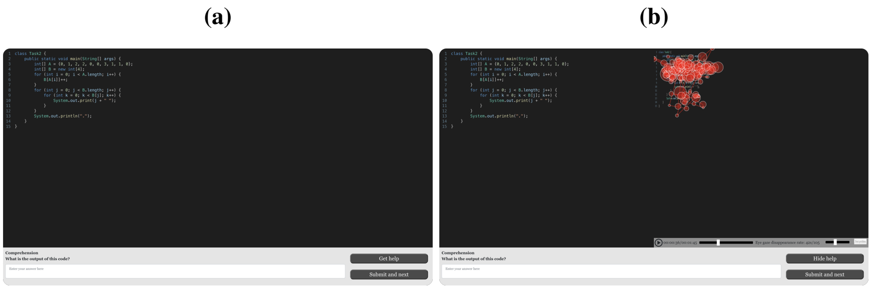
There will be no possibility to compile the code.

Help during the tasks

During the tasks, you will have the opportunity to get help in the form of an expert's eye gaze. The help can be enabled by pressing the "Get help" button. "Hide help" hides the eye gaze.

Figure 3.2: The two information pages in the system. a) is the information given before starting the pretest and b) is the information given before starting the study.

The study tasks' design was similar to the pretest but divided into three sections instead of two. Figure 3.3 shows the design of the ”on-demand”-help type. The third section contained the type of task (debug or comprehension), the question, and a text box where participants could input their answers or thoughts.



```java
class Task2 {
    public static void main(String[] args) {
        int[] A = {0, 1, 2, 2, 0, 0, 3, 1, 1, 0};
        int[] B = new int[4];
        for (int i = 0; i < A.length; i++) {
            B[A[i]]++;
        }
        for (int j = 0; j < B.length; j++) {
            for (int k = 0; k < B[j]; k++) {
                System.out.print(j + " ");
            }
        }
        System.out.println(".");
    }
}
```

Comprehension
What is the output of this code?
Enter your answer here

00:00:36/00:01:45 Eye gaze disappearance rate: 428/105

Figure 3.3: On-demand help task type. a) shows the layout when the participant has not requested help and b) shows the layout when the participant has pressed the button for help.

Figure 3.3a shows the screen when the participant has not pressed the button for showing the expert's eye-gaze, and Figure 3.3b shows the screen when the participant has pressed the button. By using a similar design as the pretest, the participants could get familiar with the system and would not have to struggle with unfamiliarity in the actual study. The design of the study tasks was almost identical for the different help types, with the most notable difference being the opportunity to turn the help section on and off in the ”on-demand”-help type.

Figure 3.4 portrays the help section that was displayed to the participants. The participants had the ability to fast-forward in the eye-gaze, pause and play and set the disappearance rate of the eye-gaze. The figure is the right hand side of Figure 3.3b. The controls shown in the figure would appear when a participant moved the pointer into the help section.

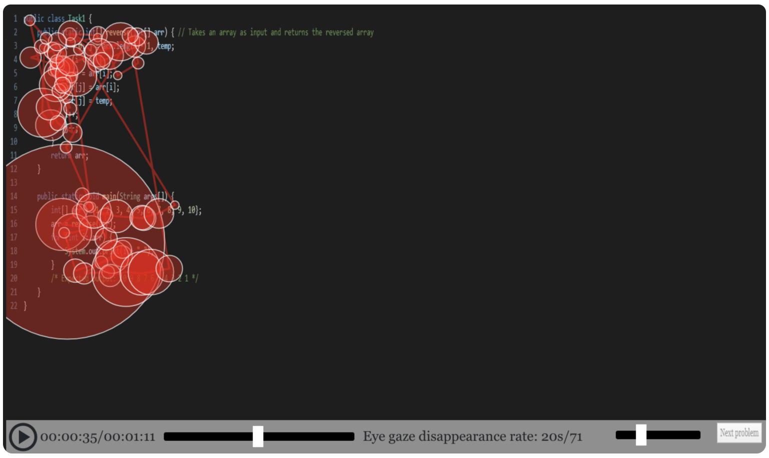
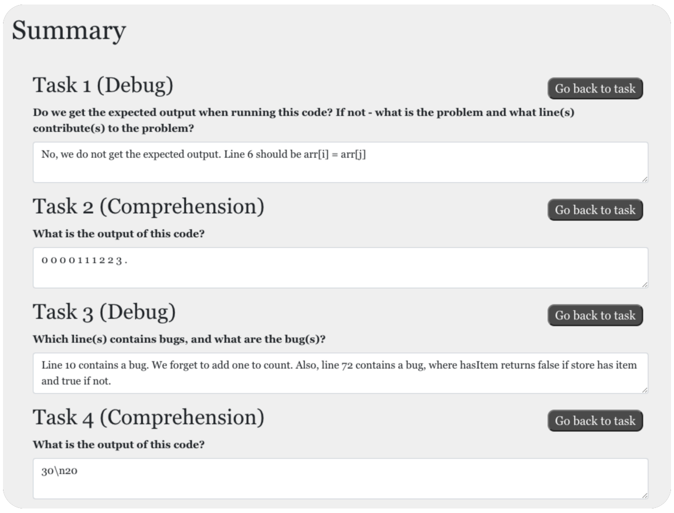


Figure 3.4: Functions available in the help section when viewing an expert's eye-gaze. The image is cropped vertically.

The system also had a summary page for the participants to review their answers before ending the study. A mockup of the summary page can be seen in Figure 3.5.



Summary

Task 1 (Debug) — Go back to task

Do we get the expected output when running this code? If not - what is the problem and what line(s) contribute(s) to the problem?

No, we do not get the expected output. Line 6 should be arr[i] = arr[j]

Task 2 (Comprehension) — Go back to task

What is the output of this code?

0 0 0 0 1 1 1 2 2 3 .

Task 3 (Debug) — Go back to task

Which line(s) contains bugs, and what are the bug(s)?

Line 10 contains a bug. We forget to add one to count. Also, line 72 contains a bug, where hasItem returns false if store has item and true if not.

Task 4 (Comprehension) — Go back to task

What is the output of this code?

30\n20

Figure 3.5: Summary page where the participants could review and update their answers.

3.1.2 Architecture

This subsection will describe the study system using a version of the 4+1 model view architecture [24].

Process view

Figure 3.6 represents the study system's run time and the components relative to its performance. It was divided into three sections; the frontend, backend, and Tobii Pro Lab software. The frontend was built using the React.js library. React is responsible for rendering the system and accepting input, validating the input, storing data, and syncing with the backend. All data input to the system was stored in the browser's session storage not to be lost by an unexpected browser refresh and to ensure the deletion of answers between participants. In addition, all user interactions were logged, such as changing tasks or playing and pausing the help video and stored in the session storage. Once a participant finished the pretest or study tasks, this data was collected from the session storage and sent to the backend for processing and storing. The backend was built using Node.js, which main focus was endpoint validation, responses, and file management. The backend checks whether the user is authenticated, sends necessary eye-gaze data to the frontend, and stores data such as logs and answers from the frontend in csvs.

Even though the Tobii Pro Lab software [25] has no interaction with the backend, it is a process that has contributed to the functionality of the study system. For example, the expert's eye-gaze data was recorded and exported through the Tobii Pro Lab software and placed in the system's backend.

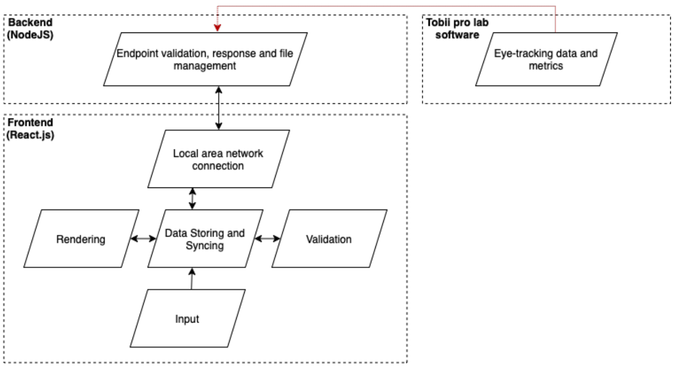

Backend (NodeJS)
Endpoint validation, response and file management
Tobii pro lab software
Eye-tracking data and metrics
Frontend (React.js)
Local area network connection
Rendering
Data Storing and Syncing
Validation
Input

Figure 3.6: Process view of the study system. Divided into backend, frontend and Tobii pro lab software.

Logical view

The app consisted of several components connected in a hierarchical order. A diagram of the app's components can be seen in Figure 3.7. A more extensive diagram version can be found in appendix A.1 in Appendix A.

`App` is the main component of the system. Every other component is being rendered inside it. In addition to being responsible for rendering the correct component based on which URL is visited, `CustomRouter` makes sure the user is redirected to the login page and that `Login` is being rendered if the user is unauthenticated. Every component rendered inside `PrivateRoute` requires the user to be authenticated. Even though it could look like it from Figure 3.7, parent components do not always contain their children. For instance, `PretestQuestions` is only rendered inside `Main` while the user is visiting the `/pretest/:id` endpoint. The green arrows in the figure indicate a relationship that always exists.

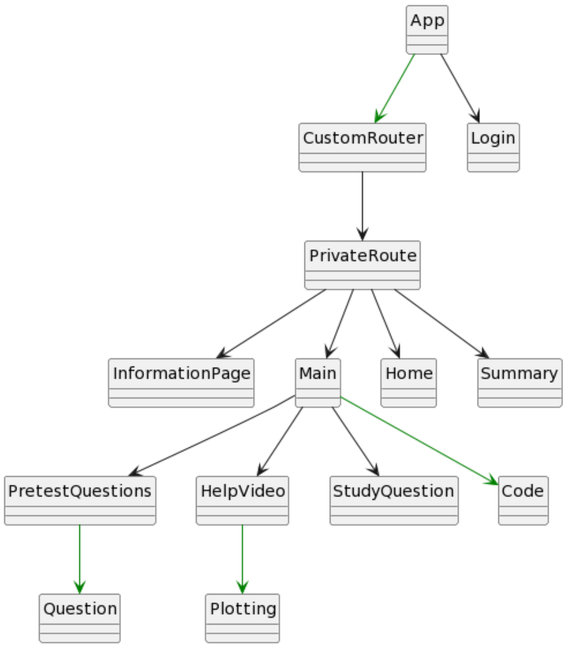


Figure 3.7: Logical view of the study system. Components are connected in a hierarchical order. Green arrows represents relationships that are always present.

Development view

This section will explain the development environment and how it is layered. As previously mentioned, the system was built using React.js frontend and Node.js backend. In addition, several libraries and frameworks were used. The main libraries used for the frontend, aside from React.js, were prism.js for code highlighting and plotly.js for plotting the expert's eye-gaze. The backend used csv-writer and csv-parser for reading and writing to csvs and the Express framework for the API. GitHub was used for code storing, and Visual Studio Code with Prettier code formatter was used as the primary IDE.

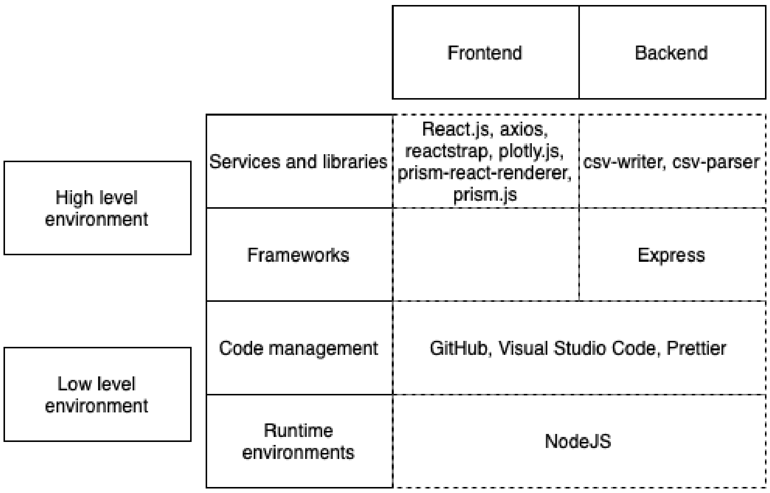


Figure 3.8: Development view of the study system. Broken down into high and low level environment.

Physical view

The system's physical components were the PC running the Tobii Pro Lab software and the eye-tracker. Since the system was running at localhost and all data stored on the local computer, no other physical components were present.

Scenarios

Figure 3.9 shows a typical system scenario. At first, a participant opens the system and is greeted with a login page if they are not previously logged in. After login, the information page in Figure 3.2a is displayed, and they have the option to start the pretest. After pressing the ”start pretest” button in Figure 3.2a, the pretest starts, and they are given a series of pretest questions such as the one portrayed in Figure 3.1. Once they have completed all pretest questions, the study information page in Figure 3.2b is presented to them, giving them information about the questions ahead and what type of help they will get. After starting the study, they will complete three debugging and three code comprehension tasks similar to that of Figure 3.3b. Once finished with the tasks, they will enter the summary page seen in Figure 3.5 and have the option to change their answers and return to the task. After delivering their answers, a ”thanks for participating”-page is shown.

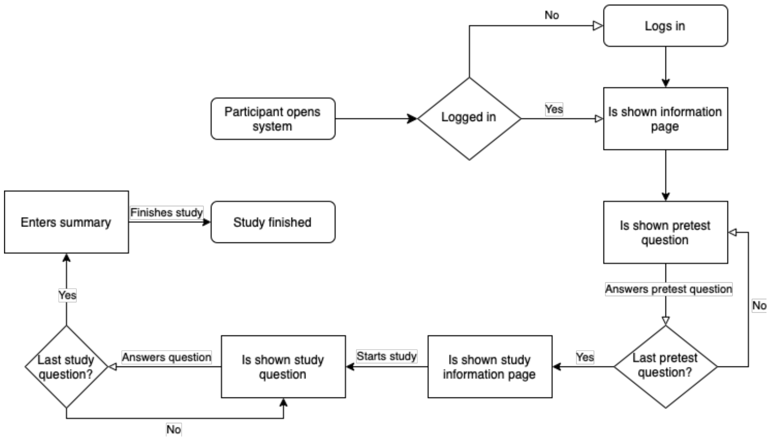


Figure 3.9: Overall flow of the study system. A participant logs in to the system and is greeted with a login page if they are not already logged in. They are then greeted with an information page informing them of the pretest. Later, they cycle through all ten pretest questions and are greeted with information about the actual study questions. Then, they start the study and cycle through six debugging and comprehension tasks. Finally, a summary page and thank you page is shown.

System testing

Before recruiting participants, the system was manually tested by the developer himself by pretending to be a participant using the system. He answered the pretest and intentionally tried to break the system by selecting multiple answers, leaving answers open, and answering incorrect and correct questions. When testing the study part of the system, which had textual answers, the developer again tried to break the system by answering every set of characters, line breaks, tabulators, and symbols. The page was refreshed multiple times to see that the answers did not disappear and disappeared after the tab was closed. The expert's eye-gaze visualization was thoroughly tested by rapidly fast-forwarding the video, intense clicking on the pause/play button, and fierce dragging the disappearance rate slider.

Finally, the summary page was tested by changing the answers and seeing that the new answer was stored once the answers were submitted.

In addition to the developer's manual testing, two peers who had not participated in the development of the system were asked to be mock participants. They came to the lab and pretended to be a participant by following the scenario shown in Figure 3.9 while using the eye-tracker. They were told to enter random characters and symbols as answers and try to use the system unexpectedly. One peer did this as a control group participant, and the other as a help group participant.

Once testing was complete and the system was proven to work as expected, the recruiting of participants began.

3.2 Research design

The study was divided into two phases, much like Stein and Brennan [7]. One expert programmer was invited to the lab in the first phase. They completed a series of Java debugging and code comprehension tasks while having their eye-gaze tracked using a Tobii Pro X3-120 eye-tracker. Their eye-gaze was recorded while doing the tasks, and their answers to the tasks were explained verbally and transcribed. Once finished, their eye-gaze data was extracted and incorporated into the system for the second phase.

Later, in phase two, novice participants were invited to the lab. The original plan was for the participants to be randomly divided into four groups to study the expert's eye-gaze exposure in a between-subject manner. However, they ended up only being divided into two groups because of difficulties recruiting participants. All participants did a pretest to test their level of programming expertise before completing the same six debug and comprehension tasks as the expert while getting the appropriate type of eye-gaze help and having their eye movements tracked with the same eye-tracker as the expert. Finally, the collected data was analyzed.

3.2.1 Participants

The expert was a TA in an introductory course to Java programming. They had both previously and alongside their studies worked professionally with Java.

All participants in the study were university students either currently attending an introductory course to object-oriented programming in Java or had attended the course less than a year ago. In addition, they had previously taken an introductory course to procedural programming in Python. For the remainder of this text, participants and novices will be interchangeably used when referring to this group. It was important to have participants who could, to some extent, understand the code snippets in the tasks so that it was possible for them to reason their way to the answer based on the help they would get. That is to say, they should preferably be in the zone of proximal development (see section 2.2). To further map the participant's level of expertise, every participant completed a pretest.

The original plan was to divide the participants into four groups, so it would be possible to study the effect of the expert's eye-gaze in a between-subject manner. The first group was planned to be a control group with no help on the tasks to study the independent variable's effect. The second group would have the expert's eye-gaze on at all times, with the ability to pause, play, move forward and backward, and set the disappearance rate of the eye-gaze. The third group were planned to have an ”on-demand”-type of help, where they had the option to turn the expert's eye-gaze on and off. Finally, the last group would have the eye-gaze visible on a particular task, based on the expert's input on which tasks they thought would benefit the participants most. For this, a total of 64 participants was needed. During the participant recruiting phase, however, it became apparent that recruiting 64 participants was not plausible due to the time restrictions of this thesis and low response from potential participants. Therefore, only 32 participants were recruited and divided into two groups; control and ”on-demand” help. The participant recruiting process is further explained in subsection 3.2.4 below.

Each participant was instructed to read through and sign an information letter and consent form before starting the study, consenting to the collection and analysis of their data. The information letter and consent form can be seen in Appendix B.

Both groups of participants did the same tasks as the expert did, with the appropriate eye-gaze visualization available, while having complete control over the eye-gaze regarding playing and pausing it, fast-forwarding, rewinding, and controlling the eye-gaze disappearance rate. The answers and eye-gaze of the participants were stored locally on the computer.

After all participants had completed the study, their eye-tracking data, provided by the Tobii Pro Lab software Tobii Pro AB [25], and answers to the tasks were analyzed to determine whether exposure to an expert's eye-gaze could help improve a novice's problem solving time and accuracy.

3.2.2 Determining programming expertise

As explained in subsection 2.3.3, it is important to have a way to measure programming experience. Even though the study performed in this thesis targeted university students that had taken an introductory programming course, it is not given that all of them were novices in Java and object-oriented programming. Therefore, a code-pretest, was used to measure their programming experience. An example pretest code and question can be seen in Figure 3.1. The pretest was inspired by tasks appearing after searching ”Java interview questions.” All pretest code snippets can be found in Appendix C.

3.2.3 Study tasks

The task types, debugging and comprehension, were planned to be studied within-subject. By doing the task type as a within-subject, we reduce errors associated with individual differences. E.g., one individual is likely to be at the same level in both debugging and comprehension. Therefore, it was planned that the different groups would have different ordering of the tasks, where two of the groups would start with a debugging task and then alternate between debug and comprehension. In contrast, the other two groups would start with comprehension and then alternate. However, since the decision to have only two groups arose during the second group's data collection, the tasks started with debugging tasks and alternated between debugging and comprehension.

All code comprehension snippets shown to the participants and experts were anonymized, meaning one could not derive the function of the code by simply reading the method names, class names, or comments. In addition, since all code were Java code snippets, questions and code included object states and their change. The inclusion of object states and their changes was done since object-oriented programming was presumed relatively new to the novices. All tasks can be seen in Appendix D. The tasks were inspired by previous exam questions and topics lectured in the object-oriented course.

3.2.4 Execution of the study

The study was held in two iterations over seven weeks at NTNU's premises. Figure 3.10 shows a timeline of the study. Participants were recruited in two iterations through an invite published on the TDT4100[3] course's Blackboard page[4], recruiting in a lecture break and by emailing all students currently taking, or had previously taken the course with an invite. In addition, a reminder email was sent to all students after one month of sending the first. In the invite, the students were given a brief description of the project and an attached information letter with more in-depth details and how their data would be treated. The information letter can be seen in Appendix B. In addition, they were given a Doodle[5] link where they could appoint themselves to a time slot to join the study. They registered themselves with their names and email address so that specific information could be sent to each individually. Furthermore, students were visited at their designated reading areas to promote the study and by leaving QR codes to scan for more information.

Once registered for a time slot, an email containing more information about the pretest and study, what they needed to bring, and where and when they should meet was sent. A reminder email was sent one day before a participant's chosen time slot in case they forgot they had volunteered to participate. In some cases, rescheduling was needed, either later that same day or another day. As mentioned, this recruiting process happened in two distinct intervals. Figure 3.10 gives a visual guide to better understanding the study's timeline. At first, students *currently* taking the introductory course to object-oriented programming were invited. In the second interval, students that had taken the course less than a year ago were invited. This was done because of problems with recruiting students currently taking the course.

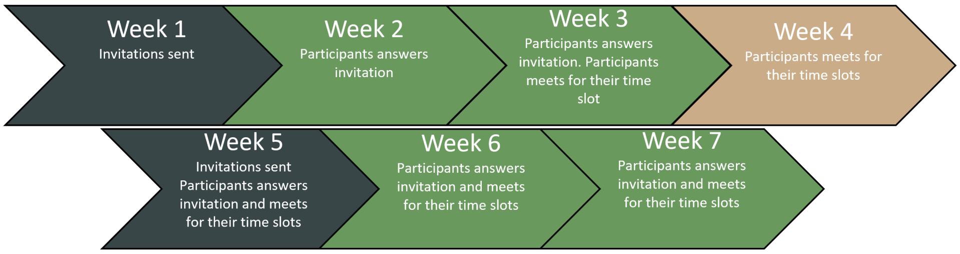

Week 1
Invitations sent
Week 2
Participants answers invitation
Week 3
Participants answers invitation. Participants meets for their time slot
Week 4
Participants meets for their time slots
Week 5
Invitations sent
Participants answers invitation and meets for their time slots
Week 6
Participants answers invitation and meets for their time slots
Week 7
Participants answers invitation and meets for their time slots

Figure 3.10: Timeline of the study. The study span over seven weeks, and participants were recruited in two intervals.

When the participants met for their assigned time slot, they were instructed to read through a physical copy of the information letter sent by email and sign the consent form. Then, information about how the session would go and the pretest and tasks

[3]`https://www.ntnu.edu/studies/courses/TDT4100`

[4]Blackboard (`https://www.blackboard.com`) is NTNU's answer to Canvas, or It's learning

[5]`https://doodle.com/en/`. A tool for scheduling meetings

were given verbally. In addition, all information that can be seen in Figure 3.2 was given verbally. Furthermore, the participants were told that the observer would be sitting beside them in complete silence during the whole session, working on their own tasks and that the observer would not pay any attention to what answers the participants gave to the tasks; they would only keep an eye on the time and how far along with the tasks the participant had gotten. They were informed that they had one hour and 15 minutes to complete both the pretest and study tasks, but should try to not spend more than 15 minutes on the pretest. They were also informed that they were allowed to ask the observer questions but that the observer might not be able to answer. E.g., questions regarding the system could be answered, but not questions directed to the code. Following this, the study formally started with a calibration of the eye-tracker. From this point on, the study followed the scenario given in subsection 3.1.2, with the pretest followed by the study tasks.

Once the help group started the first study task, the observer asked them to open the help section so that they could explain how it worked. The observer showed the participants the fast-forwarding, pause and play, and the disappearance rate option. They also explained that the larger the circles were, the more time the expert had spent looking in that area. All answers the participants gave to the pretest and study tasks were stored in the browser's session storage for easy access to the data and to ensure the data would not be lost at an unexpected browser refresh. During the entire study, all user interactions, such as changing tasks, playing, or pausing the help video, were logged and stored in the browser's session storage.

All participants were given a notebook to note their thoughts and keep track of the code during the problem solving. This notebook was not used or even looked at by the observer, and every participant's page was discarded between participants. Not all participants used this notebook.

During the study, the observer sat in silence beside the participants. Occasionally, the participants had some questions regarding the tasks, which were answered and clarified. It should be noted that this does *not* mean the observer gave any hints or help directly linked to the tasks, but clarification about the questions in the tasks and how the study system worked. Before the participants began the study tasks, they were greeted with the information in Figure 3.2b. Once again, this information was given verbally when the participants got to this screen. In addition, all participants were informed that they could assume the comprehension tasks were bug-free. Therefore, the code would compile and give the desired output if one tried to compile it.

Once finished, the participants were given a 200NOK[6] "Midtbyen" gift card[7]. All participants also entered a lucky draw for a 500NOK gift card that was drawn after the study ended. They were then asked about the tasks - what did they think about them? Some participants also wanted a brief review of some of the tasks and asked the observer to explain some concepts such as Java streams[8] and ternary operators. Some participants spent the whole hour and fifteen minutes, while others finished within forty minutes.

After all participants had completed the study, their answers and eye-tracking data were analyzed.

3.3 Analysis

This section presents the analysis of the data collected in the study described above. It starts with preprocessing of the raw data in subsection 3.3.1 and a description of the final datasets in subsection 3.3.2. Section 3.3.3, subsection 3.3.4 and subsection 3.3.5 describes the analysis done on the processed data.

All preprocessing of the data was done in Python, and the analysis was conducted in RStudio version 2021.09.0 Build 351 for Windows 11.

3.3.1 Preprocessing

As mentioned in subsection 3.1.2, all participant answers and logs were stored locally on the computer. Before analyzing the collected data, it had to be processed. The pretest and study data had to be graded, and the eye-gaze data had to be ordered better than the raw collected data.

Pretest

As described in subsection 3.2.2, every participant did a pretest to test their level of expertise. The raw pretest data contained the fields `subjectId`, `username`,

[6]Roughly $23 at time of writing (2022-03-22)

[7]`https://midtbyen.no/midtbykortet-gavekort-for-trondheim-sentrum`. A gift card that can be used in 250 stores and cafes in Trondheim centrum

[8]`https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html`. As the study progressed, more Java concepts were lectured. As a result, some participants had more knowledge of streams than others

`pretest.`N`-1` and `pretest.`N`-1_correct` for $1 \leq N \leq 10$. Some example rows can be seen in Table 3.1.

Table 3.1: Example rows for the raw pretest data. The answers are not associated with the username.

subjectId	username	pretest.1-1	...	pretest.10-1_correct
43	cleararagog	2	...	0
45	bossyhogwarts	3	...	0

Since the pretest was a multiple choice quiz and each question had only one correct answer, the grading process for it was automated by the Python script found in section E.1, in Appendix E.

Each participant's answers were iterated and checked with the correct answer for the respective task. A cumulative sum was created for each participant and added into a new field `no_correct`. In addition, the fields `pretest.`N`-1_participant_correct`, one if the participant had anwered the question correct and zero if not, and `helpType` were created. The final dataset contained these fields, in addition to the ones in Table 3.1.

Study

In addition to processing the pretest data, the study tasks data had to be preprocessed as well. The raw study tasks data had the fields `subjectId`, `username` and `studyTask`N, where $0 < N \leq 6$. Example rows for the study tasks data can be seen in Table 3.2.

Table 3.2: Example rows for the study tasks data. Answers are not associated with the username.

subjectId	username	studyTask1	...	studyTask6
43	cleararagog	”No, line 6 should be arr[i] = arr[j]”	...	”70”
45	bossyhogwarts	”lines 6 and 7 should add to the array temp not equal”	...	”A integer of value I cannot find”

As the study tasks had written answers, they had to be graded manually. The answers were anonymized to prevent grading bias. This means the grader did not know for which participant they were grading. Only which task and the answer that was given. This was done through the script found in section E.2, in Appendix E. Each answer was given a grade based on its correctness. For the debugging tasks, this meant one point per bug identified and boolean correct or wrong for the comprehension tasks, for a total of 8 points. The resulting dataset for the study contained the fields `studyTaskN_points`, for $1 \leq N \leq 6$, `subjectId`, `username`, `helpType` and `no_points`.

Eye-gaze

The processing of the eye-gaze dataset was more extensive than the pretest and study data. At first, five large AOIs were created for each of the six study tasks; `TaskX.Code`, `TaskX.TaskType`, `TaskX.Question`, `TaskX.Help` and `TaskX.HelpButton`. Later, AOIs for each code line within each of the six tasks were also created. The Tobii pro lab software [25] offers many metrics to be extracted alongside each AOI, such as fixations and saccades (see section 2.1), key presses and mouse clicks, and more. For each AOI, the software calculates the total number of fixations and saccades that happened inside the given AOI and the total duration of fixations and saccades in each of the AOIs. For a more extensive list of metrics the software calculates, see Tobii Pro AB [26]. A total of two datasets were extracted from the Tobii pro lab software. One contained the metrics found in Tobii Pro AB [26, sec. 10.5.4], and one contained raw eye-gaze data alongside metadata and AOI hits. All features in the data dataset can be found in Tobii Pro AB [26, sec. 11.3].

Unfortunately, the Tobii software had trouble automatically detecting which eye-tracking data belonged to which task and AOI, which caused the exported data to be incorrect. This meant that the data had to be manually adjusted. Luckily there was an option within the software to manually set the start and end time of the AOIs. This was done for every task for each participant by analyzing their recordings and manually identifying the exact start and end frame of the tasks and AOIs. The exported data then correctly represented the metrics, e.g., the number of fixations in `Task1.Code`.

When the AOIs for the code lines were constructed, they were manually created for one participant by selecting and annotating each code line in the recording. All AOIs were then exported separately for each participant, formatted as JavaScript Object Notation (JSON) [27]. Note that, at this point, only one of the exported files contained the AOIs for the code lines. The file with the code line AOIs was then used to extract the coordinates of the code line AOIs, which was the same for all participants. Each extracted participant's AOI file was then modified to include the AOIs for the lines by using the respective participant's `TaskX.Code` AOI as a reference for the start and end frame for each new AOI. The code written for this can be seen in Lindberg [28]. Once this was done for all participants with no registered AOIs for the code lines, the files were uploaded to their respective recordings in the Tobii software, and the metrics and raw eye-gaze data were extracted.

The extracted datasets were further processed by renaming each of the AOI columns to more friendly names than what Tobii automatically generates for them. In addition, a new column `Task` was added to the data dataset, by utilizing the `start of interval`, `duration of interval` and `TOI` columns in the metrics dataset. Again, refer to Tobii Pro AB [26, sec. 10.5.4] for an explanation of the different features. Both the code for renaming the AOIs and the addition of the new column can be seen in Lindberg [28].

3.3.2 Datasets

Due to the large amount of data and limited computing power, the exported datasets from the Tobii software were split into 2×32 separate files. One metric file for each participant and one data file for each participant. For the remainder of this thesis, the 32 separate metric files will be referred to as one dataset named the metric dataset and the 32 separate data files as the data dataset. When conducting the analysis, the metrics and data datasets were the main data sources. However, one more dataset was created. This last dataset contained 32 participants (ten females) and was created by merging the preprocessed pretest and study datasets and the eye-gaze dataset described in subsection 3.3.1 above. In addition, multiple new features were included in the final dataset. These new features can be seen in Table 3.3 below.

The first three features in Table 3.3 are groups the participants were placed in based on their scores and help type. The decision to split the participants into groups of novice and expert expertise was inspired by Najar et al. [4] that found a significant difference in reading behaviors for the two groups. Furthermore, by studying their facial expressions, Sharma et al. [19] found differences in high and low performers. Therefore, it was decided to split the participants in this study into high and low performers, to investigate the difference in their reading behaviors. The splits for the first two features in Table 3.3 were done on the median of the respective scores. The dataset was left with 16 high and 16 low expertise participants and 17 and 15 high and low performers, respectively. The split for the help group was merely a change in the already present `helpType` feature from integer to strings. The rest of the features in Table 3.3 were created by summing over each participant's respective metrics for all study tasks.

Table 3.3: New features introduced in the processed dataset with description.

Feature	Description
Expertise.Group	One of ”High” and ”Low”. High if participant scored higher than the median pretest score on their pretest
Performance.Group	One of ”High” and ”Low.” It was high if participants scored higher than the median study tasks score in their study tasks.
Help.Group	One of ”help” and ”ctrl,” based on their help type.
Total_number_of_code_fixations	Row sum of the already present features Number_of_fixations.TaskN.Code, for $1 \leq N \leq 6$
Total_duration_of_code_fixations	Row sum of the already present features Total_duration_of_fixations.TaskN.Code, for $1 \leq N \leq 6$
Total_number_of_help_fixations	Row sum of the already present features Number_of_fixations.TaskN.Help, for $1 \leq N \leq 6$
Total_duration_of_help_fixations	Row sum of the already present features Total_duration_of_fixations.TaskN.Help, for $1 \leq N \leq 6$

3.3.3 Difference in study and pretest scores

Since this thesis investigates what effect exposing **novice** programmers to an expert’s eye-gaze have on their comprehension and debugging abilities, it is essential to make sure the participants actually were novices and that they all were on the same level of expertise. This is important so that the study results do not become biased. E.g., if all participants that got to see the expert programmer’s eye-gaze were all experts, they would more likely be able to answer the tasks correctly than the less able participants. Therefore, the pretest scores were analyzed using a two-sample independent Student’s t-test to check for expertise level differences.

Furthermore, the most apparent analysis would be to investigate differences in study scores between the control and help groups. Therefore, a two-sample independent Student’s t-test was performed on the study scores.

Another way to reach the goal of this thesis was to answer the question, ”What are the differences between high and low expertise novice programmers when exposed to an expert programmer’s eye-gaze?”. As explained in subsection 3.3.2, the participants were split in to high and low expertise based on their pretest scores. Following this split, a difference in their study scores was investigated with a two-sample independent Student’s t-test. The same was done to investigate **RQ3.3** regarding high and low performers.

3.3.4 Number of fixations and total duration of fixations

As Crosby and Stelovsky [3] found that highly experienced subjects recognized and spent more time concentrating on meaningful areas, the difference between the participants with regards to identifying meaningful areas after being presented with the expert’s eye-gaze was analyzed. Moreover, Najar et al. [4] found a significant difference in high and low expertise participants with regards to using the help presented. Therefore, the total number of fixations on each of the most meaningful areas and their duration were analyzed.

Table 3.4: Example rows of eye movement types from the data dataset with their respective X and Y coordinates and recording timestamps. Colored in order to show that the fixation eye movement type was kept while saccades were discarded.

Recording timestamp	Eye movement type	X	Y
100000000	Fixation	346	525
100008000	Fixation	346	525
100011634	Fixation	346	525
100012000	Saccade	nan	nan
100015000	Saccade	nan	nan
100017000	Fixation	569	458
100020957	Fixation	569	458
100025697	Fixation	569	458

At first, the duration for each fixation in the data dataset was calculated. Table 3.4 shows an example of rows from the dataset. Consecutive rows with `Eye movement type` fixation and identical `X` and `Y` values (displayed as green rows) are events from the same fixation. For the most part, such consecutive rows are separated by `Eye movement type` saccade rows (displayed as gray rows) and are considered different fixations. In the table, there are two fixations, one fixation on the point $(346, 525)$ on the screen and one on the point $(569, 458)$. The two fixations are separated by one saccade. The `Recording timestamp` column is simply microseconds since the recording started. The difference in `Recording timestamp` values between the first and last occurring rows in such a consecutive series was calculated to find the duration of a fixation. In Table 3.4 that would be $100011634 - 100000000 = 11634\mu s$ for the first fixation and $100025697 - 100017000 = 86974\mu s$ for the second fixation. The code for calculating fixation duration can be seen in Lindberg [28].

To better compare participants' fixation duration, the duration were normalized. This was done by dividing each duration by the duration of the task the fixation occurred. The task start was subtracted from the task end to find the task duration for the control group. This was done for the help group as well. However, the task duration would then include the time spent looking at the help, which would incorrectly represent the help group's task duration. Therefore, to get the correct task duration for the help group, every fixation duration on the help section was subtracted from the total task duration. E.g., if a participant's total task duration was 100 seconds, and the participant had consulted the help section for 10 seconds, the new task duration would be 90 seconds. This new task duration was used to normalize the fixation duration for the help group. The fixation duration on each AOI was then summed together to get the total duration for each of the AOIs.

Finally, two new dataframes were created. One contained the total fixation duration on each AOI, alongside which AOI was fixated. The second contained individual fixations, their duration, and which AOI was fixated. The difference in fixation duration on the lines with bugs between groups was tested with Student's t-tests and Kruskal-Wallis rank sum tests. Furthermore, the number of fixations on a specific AOI could be calculated by counting occurrences of the given AOI in the second new dataframe. A difference in the number of fixations on a given AOI was then tested with Student's t-tests and Kruskal-Wallis rank sum tests.

3.3.5 Distance to lines with bugs

To further investigate an expert's eye-gaze effect on novice programmers, distances of the first fixation after a participant consulted the help section was analyzed.

At first, series of fixations such as the ones shown in Table 3.4 with the fixation being in one of the help sections was identified. Then, the first subsequent fixation, which was not located in the help section, was identified. Following this, the distance d from the subsequent fixation to the bugs in the respective task was calculated using the formula in Equation 3.1 below. In the formula, min_x, max_x, min_y and max_y refer to the maximum and minimum X and Y pixel coordinates of the bug AOI-rectangle, and F_x and F_y is the coordinates for the subsequent fixation. The distance was measured in pixels.

$$d = \sqrt{\max(min_x - F_x, 0, F_x - max_x)^2 + \max(min_y - F_y, 0, F_y - max_y)^2} \tag{3.1}$$

The distances to each bug on the respective task and the AOI that was first hit after consulting the help section were added to a new dataframe. If both the code section and a line AOI were hit, the line AOI was added. The percentage of all first fixations after consulting the help section that was on a line with bug was then calculated by dividing the number of first fixations on lines with bugs by the total number of first fixations after consulting the help. This was done both for individual tasks and total for all tasks. The correlation between these percentages and study scores for the help group was then analyzed with a Pearson correlation.

Another dataset was created regarding the distance to lines containing bugs. This time, the distances were grouped as percentiles from the bug. The line furthest away from the bug was labeled 100% distance from the bug, and the line with the bug was labeled 0% away. Every other line was either 25%, 50% or 75% distance from the bug. The already created distance-dataframe described in subsection 3.3.4 was used to generate this new dataset. By using the coordinates of the AOI that was fixated, the distance d to the bug was found by using Equation 3.2 below, where min_{xn}, min_{yn}, max_{xn} and max_{yn} is the minimum and maximum x and y values of the bug AOI and the line AOI. The values d_{xn} and d_{yn} are measures of the shortest distance between the two AOIs on the x and y-axis, respectively, and makes sure the distance is 0 if the AOIs overlap either horizontally or vertically. Finally, d is the distance between the two AOIs. The formula is inspired by the Euclidean distance between two points in space.

$$\begin{aligned} d_{x1} &= \max(min_{x1} - min_{x2}, 0, min_{x2} - max_{x1}) \\ d_{x2} &= \max(min_{x2} - min_{x1}, 0, min_{x1} - max_{x2}) \\ d_{y1} &= \max(min_{y1} - min_{y2}, 0, min_{y2} - max_{y1}) \\ d_{y2} &= \max(min_{y2} - min_{y1}, 0, min_{y1} - max_{y2}) \\ d &= \sqrt{\min(d_{x1}, d_{x2})^2 + \min(d_{y1}, d_{y2})^2} \end{aligned} \tag{3.2}$$

Once the distance was found, it was divided by the distance of the line furthest away from the bug to obtain the percentage distance from the bug. Then, the fixation was grouped as either 0%, 25%, 50%, 75% or 100% distance based on the constraints shown in Table 3.5.

Table 3.5: Percentage distance group constraints.

Group	**Constraint**
0%	$[0, 0]$
25%	$(0, 25]$
50%	$(25, 50]$
75%	$(50, 75]$
100%	$(75, 100]$

After each fixation had been grouped, the difference between conditions, expertise groups, and performance groups was investigated using Student's t-tests and Kruskal-Wallis rank sum tests.

Chapter 4

Results

This chapter contains the results of the analysis described in section 3.3. It is divided into three subsections; one for the conditions, section 4.1, one for the expertise groups, section 4.2 and one for the performance groups, section 4.3. Each section is further divided into the same subsections which presents the same result for each of the three groups. For instance, subsection 4.1.3 presents results from the number of fixations analysis explained in subsection 3.3.4 for the conditions, subsection 4.2.2 for the expertise groups and subsection 4.3.2 for the performance groups.

As mentioned in section 3.2, this study used the Tobii Pro X3-120 eye-tracker. A mean percentage of 68.34% gaze samples was recorded across participants. According to [29], the Tobii Pro X3-120 eye-tracker is not 100% accurate. In this study, the mean accuracy across all participants was 53 pixels. The mean and standard deviation for pretest and study scores for individual tasks and the number of fixations and duration in individual AOIs can be found in section F.2 in Appendix F. All results are produced by analyzing the collected data in RStudio version 2021.09.0 Build 351 for Windows 11.

4.1 Conditions

This section will present analysis results for the two conditions, control and help.

4.1.1 Pretest scores

As explained in subsection 3.3.3, it was essential that all participants were on the same expertise level to reduce bias in the results of the study. Figure 4.1 visualizes the pretest scores of the control and the help group.

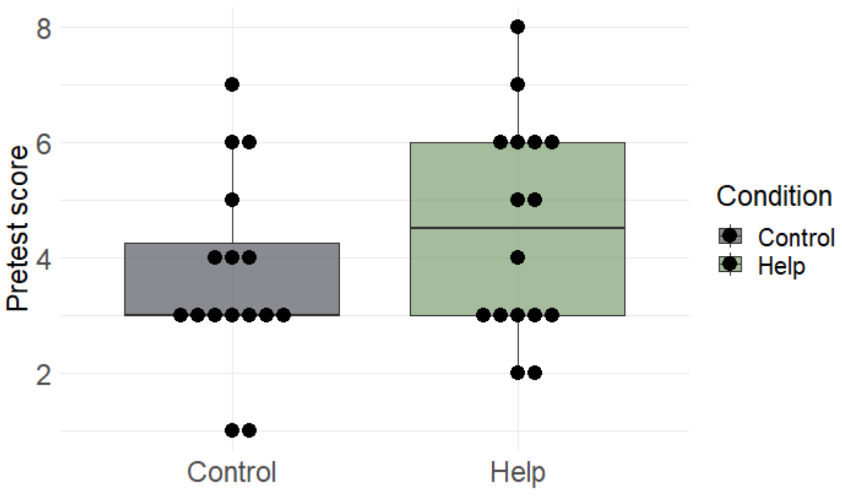


Figure 4.1: Boxplot of number of correct answers on the pretest for the two conditions; control and help.

A Shapiro-Wilk normality test was performed on the total number of correct answers on the pretest for each participant, which did not present evidence of non-normality for the control group ($W = 0.90619$, $p = .101$), nor for the help group ($W = 0.9134$, $p = .132$). Furthermore, a Breusch-Pagan Test was performed to check for heteroscedasticity. The test presented a result of non-heteroscedasticity ($BP(1) = 0.34154$, $p = .5589$), meaning one can assume equal variances in the two conditions. Finally, as the data contained two independent groups and showed signs of normality and non-heteroscedasticity, a two-sample independent Student's t-test with significant level $\alpha = .05$ was performed, which showed no significant difference in number of correct pretest answers between the control group ($M = 3.6875$, $SD = 1.662077$) and the help group ($M = 4.5$, $SD = 1.861899$)

conditions; ($t(30) = -1.3022$, $p = .2028$).

4.1.2 Study scores

With no significant difference in the number of correct pretest answers, one could assume that all participants were at the same expertise level and that the rest of the results are not biased towards expertise. Since one of the main goals of this thesis was to investigate what kind of effect exposing novice programmers to an expert programmer's eye-gaze has on their comprehension and debugging abilities, one should investigate the difference in the conditions' scores on the study tasks.

Figure 4.2 shows the average of the conditions' scores for each of the tasks and Figure 4.3 displays a boxplot of the total study scores for the two conditions.

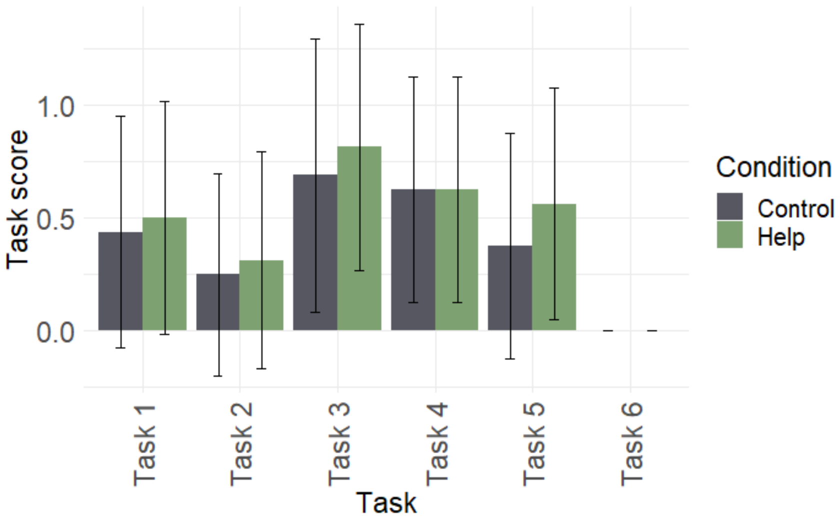


Figure 4.2: Average task scores for the two conditions; control and help. Error bars represent the standard deviation for each task and group.

A Shapiro-Wilk normality test was performed on the study scores for each participant, which did not present evidence of non-normality for the control group ($W = 0.92604$, $p = .2109$), nor for the help group ($W = 0.92599$, $p = .2105$). Therefore, a Breusch-Pagan Test was once again performed, to test for heteroscedasticity. The test yielded a result of non-heteroscedasticity ($BP(1) = 0.44862$, $p = .503$), meaning one could assume equal variances for the two conditions. Finally, a two-sample independent Student's t-test with significance level $\alpha = .05$ was performed, which showed no significant difference in study scores between the

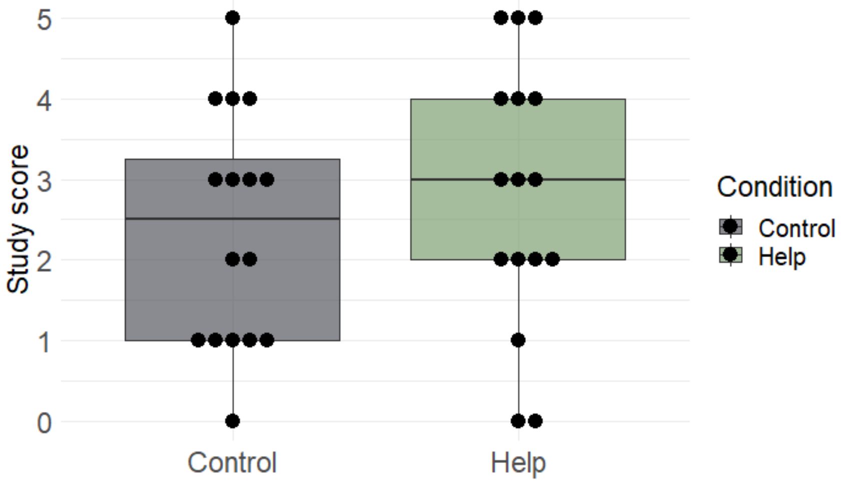


Figure 4.3: Boxplot of the study scores for the two conditions; control and help.

control group ($M = 2.3750$, $SD = 1.454877$) and the help group ($M = 2.8125$, $SD = 1.641899$) conditions; ($t(30) = -0.79772$, $p = .4313$). More extensive testing was performed as well, see subsection F.1.1 in Appendix F.

4.1.3 Number of fixations on lines with bugs

As demonstrated by Crosby and Stelovsky [3], highly experienced subjects are more likely to recognize and spend more time concentrating on meaningful areas in code. Therefore, this section will present results concerning time spent and fixations on lines containing bugs in the code, categorized as meaningful areas, as it might indicate a difference in expertise between the two conditions.

Figure 4.4 shows a boxplot of aggregated fixations on all bugs for the two conditions.

A Shapiro-Wilk test was performed on the total number of fixations on all bugs for each of the conditions. The test showed no sign of non-normality for the control group ($W = 0.95609$, $p = .5918$) or the help group ($W = 0.93356$, $p = .2774$). Furthermore, a Breusch-Pagan test did not show evidence of heteroscedasticity between the two groups ($BP(1) = 0.77078, p = .38$). Therefore, a Student's t-test with significance level $\alpha = .05$ was performed to test for difference in total number of fixations on all bugs. The test showed no significant difference between the control group ($M = 202.75$, $SD = 128.2152$) and the help group ($M = 230.25$,

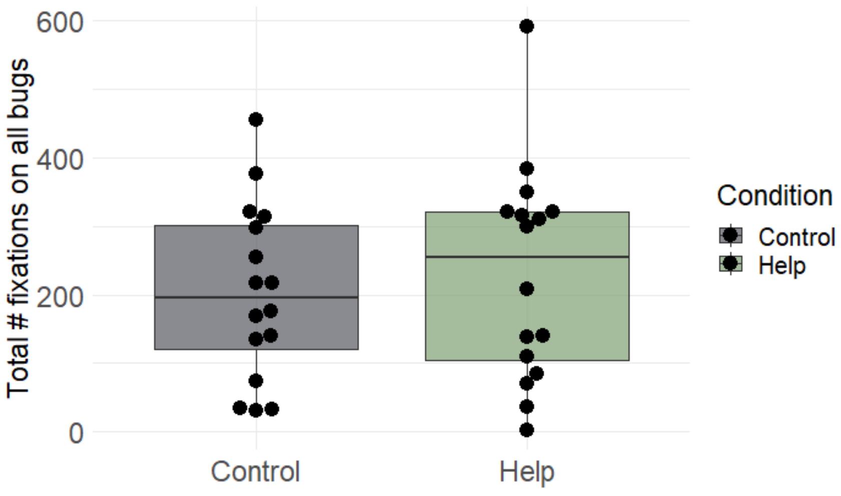


Figure 4.4: Boxplot of total number fixations on all bugs for the two conditions; control and help.

$SD = 157.1312$) conditions; ($t(30) = -0.5424$, $p = .5916$).

Even though there was no difference in number of fixations overall on all lines with bugs between the conditions, there could still be a difference in the number of fixations on individual lines with bugs. Figure 4.5 shows a boxplot of the total number of fixations on the second bug in task 5.

A Kruskal-Wallis rank sum test was performed to test for difference in total number of fixations on the second bug of task 5. The test showed a significant difference between the control group ($M = 25.8125$, $SD = 28.72216$) and the help group ($M = 44.46667$, $SD = 24.75557$) conditions; ($\chi^2 = 4.9059$, $df = 1$, $P = .02676$), meaning the help group had a significantly higher total number of fixations on the second bug of task 5 than the control group. Tests and results for the other bugs can be seen in subsection F.1.2 in Appendix F.

4.1.4 Time spent on lines with bugs

Figure 4.6 shows a boxplot of total normalized fixation duration on all bugs for the two conditions.

A Shapiro-Wilk test was performed to test for normality in total fixation duration on all bugs for the two conditions. The test showed no signs of non-normality

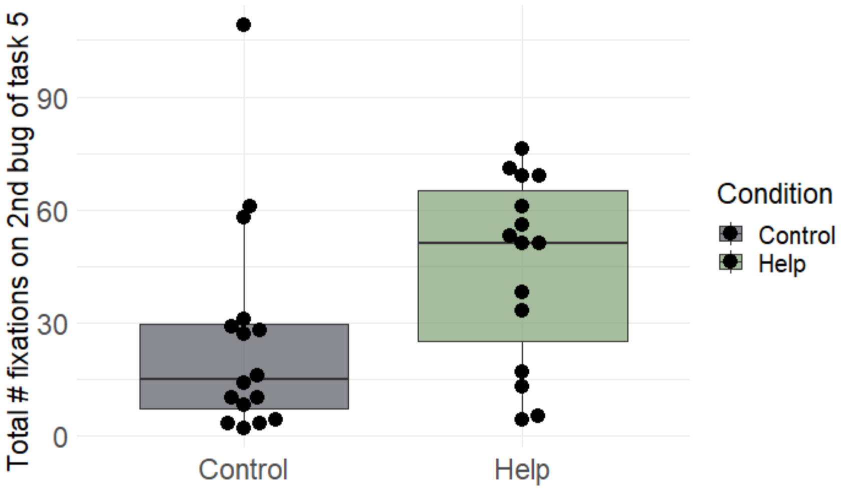


Figure 4.5: Boxplot of total number of fixations on the second bug in task 5 for the two conditions; control and help.

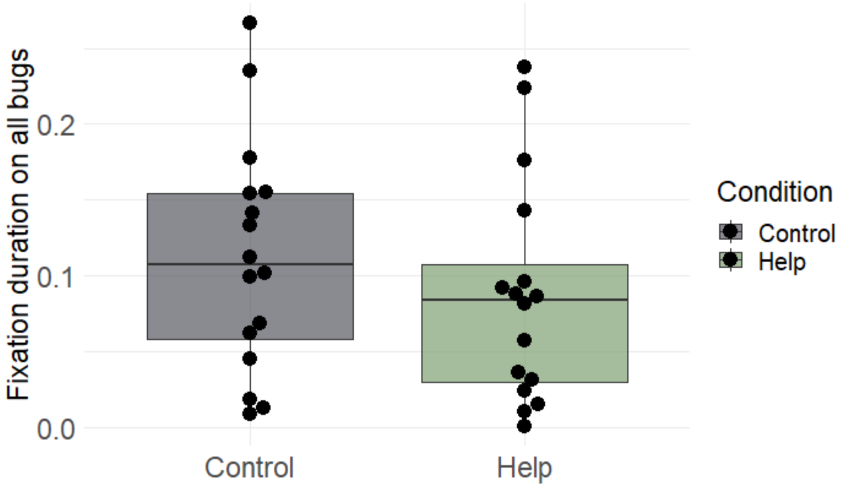


Figure 4.6: Boxplot of total normalized fixation duration on all bugs for the two conditions; control and help.

in total duration for the control group ($W = 0.95598$, $p = .5898$), nor the help group ($W = 0.8967$, $p = .07117$). Since a Breusch-Pagan test showed no signs of heteroscedacity between the two groups ($BP(1) = 0.019879$, $p = .8879$), a two-sample independent Student's t-test with significance level $\alpha = .05$ was performed to test for difference in total fixation duration between the control group ($M = 0.1121$, $SD = 0.07596$) and the help group ($M = 0.0877$, $SD = 0.0737$). The test showed no significant difference in total normalized fixation duration on the bugs, conditions; ($t(30) = 0.92417$, $p = .3628$)

4.1.5 Distance to lines with bugs

Figure 4.7 shows two diagrams representing number of fixations in different percentages away from the bugs on the tasks. Figure 4.7 shows the average number of fixations in the different percentages across all tasks and bugs, and Figure 4.7b shows a boxplot of total number of fixations in the 100 percentile on task one for the two conditions.

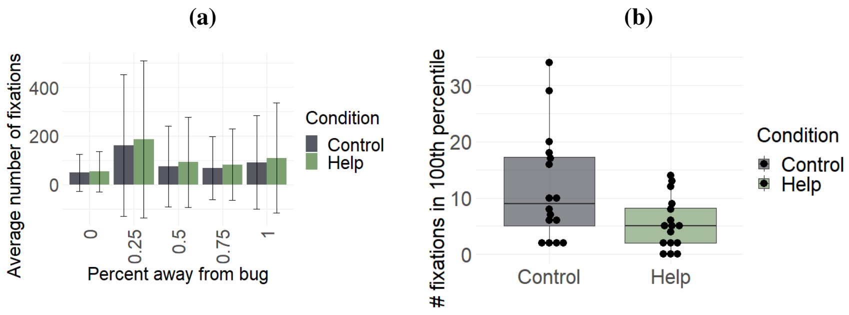


Figure 4.7: Number of fixations in percentile from the bugs for the two conditions; control and help. **a)** shows a histogram of mean and standard deviation for average number of fixations in the different percentiles from the bug. **b)** shows a boxplot of the number of fixations in the 100 percentile on task one for the two conditions.

Tests and results for all percentiles and all tasks can be seen in subsection F.1.4 in Appendix F. A Kruskal-Wallis rank sum test was performed to test for difference in number of fixations in the 100 percentile on task one between the control and help group. The test showed a significant difference between the control group ($M = 11.8125$, $SD = 9.779$) and the help group ($M = 5.4375$, $SD = 4.618$), conditions; ($\chi^2 = 4.4322$, $df = 1$, $P = 0.03527$), meaning the control group looked more on the lines furthest from the bugs on task one than the help group.

Figure 4.8 shows a correlation plot between the percentage of first fixations after looking at the help section that was on a line with a bug and the study scores for the help group.

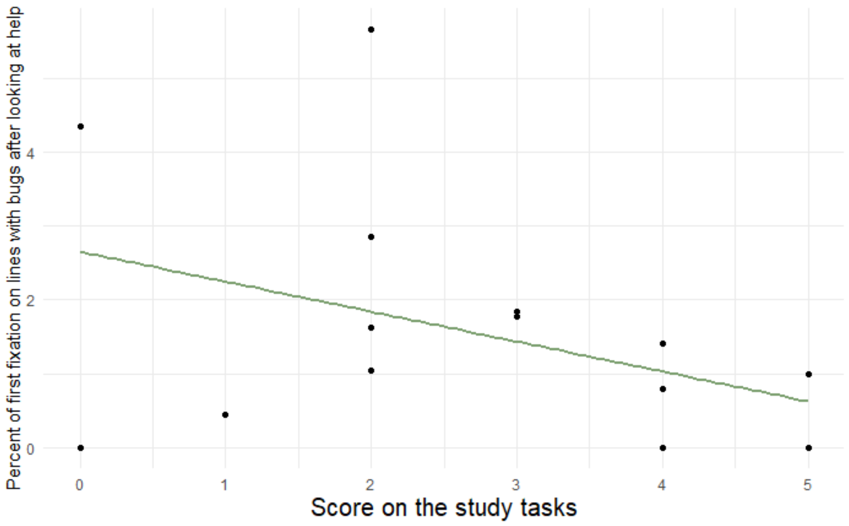


Figure 4.8: Correlation on the score of the study task as a function of the number of first fixations after looking at help that was on a line containing a bug.

From the plot in Figure 4.8, one can see a negative relationship between the two variables. A Pearson correlation was calculated which showed no significant correlation between percentage of first fixation after looking at the help section that was on a line containing bug and the study scores ($r(13) = -0.41460, p = .1244$).

4.1.6 Time spent on tasks

Figure 4.9 shows the average time in seconds the control and help group and the expert spent on each of the tasks.

A Kruskal-Wallis rank sum test was performed to test for difference in average time spent per task between the control group and help group. The test showed a significant difference between the control group ($M = 365.0905$, $SD = 235.1134$) and the help group ($M = 428.3222$, $SD = 248.5606$) conditions; ($\chi^2 = 3.9688$, $df = 1$, $P = .04635$), meaning the help group spent significantly more time per task than the control group.

Figure 4.10 shows the average difference in task durations between the conditions and the expert.

A Kruskal-Wallis rank sum test was performed to test for a difference in the time difference between the expert and the two conditions; control and help. The test

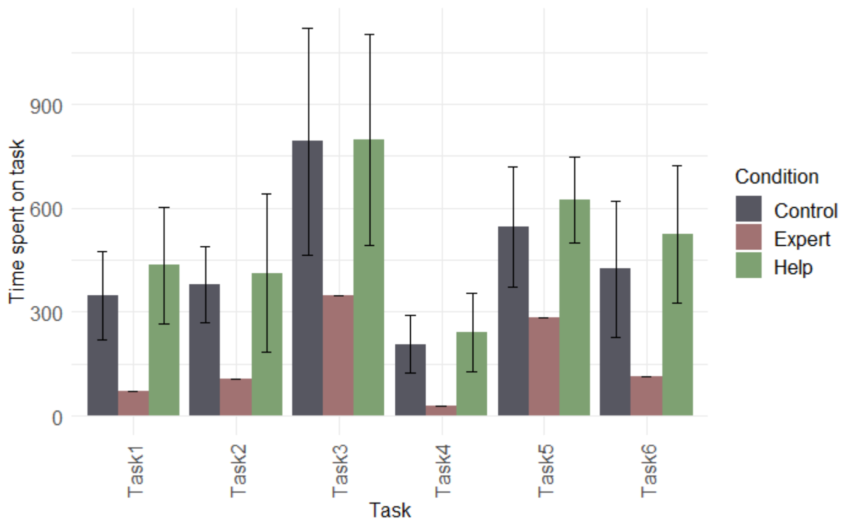


Figure 4.9: Average time spent per task in seconds for the two conditions, control and help, and the expert. Error bars represent the standard deviation for each task and condition.

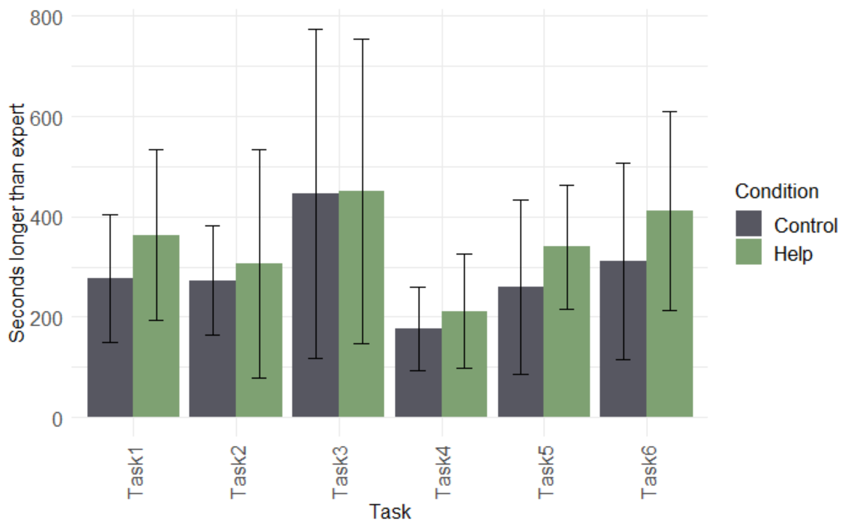


Figure 4.10: Average time difference, in relation to the expert, spent on the tasks for the two conditions; control and help. Error bars represent the standard deviation.

showed a significant difference between the control group ($M = 206.4441$, $SD = 195.3654$) and the help group ($M = 269.1244$, $SD = 204.1006$) conditions; ($\chi^2 = 5.5633$, $df = 1$, $P = .01834$), meaning the help group on average spent significantly more time per task than the expert compared to the control group.

4.2 Expertise groups

This section will present analysis results for the expertise groups.

4.2.1 Study scores

Having established no significant difference in study scores between the conditions, further analysis was done by splitting the participants into different expertise groups to investigate if the expert's eye-gaze had any effect between the different levels of expertise. The participants were, as described in subsection 3.3.2, labeled as either "high" or "low" expertise based on their score on the pretest. The split resulted in 16 high expertise participants and 16 low expertise.

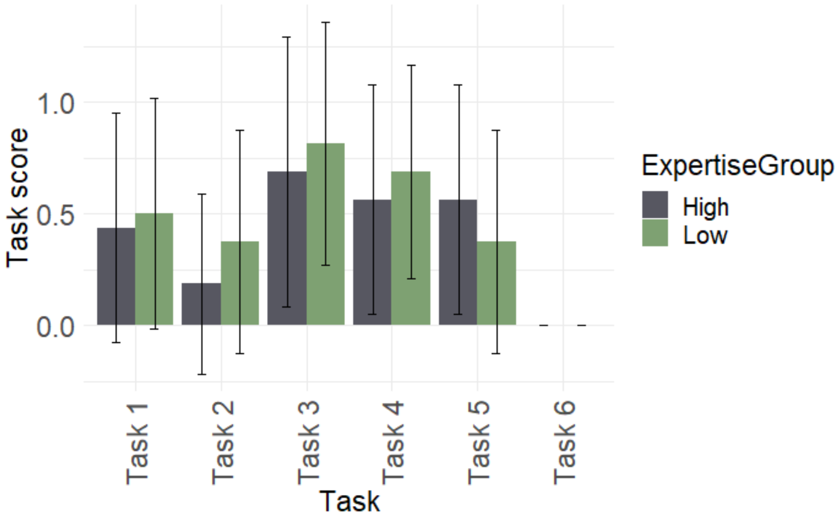


Figure 4.11: Average task scores for the two expertise groups; high and low. Error bars represent the standard deviation for each task and group.

Figure 4.11 shows the average scores of each expertise group for the study tasks.

Once again, a Shapiro-Wilk test was performed to test for normality in the data. The test did not present evidence of non-normality for the low expertise group ($W = 0.90298$, $p = .08972$), nor for the high group ($W = 0.91156$, $p = .1233$).

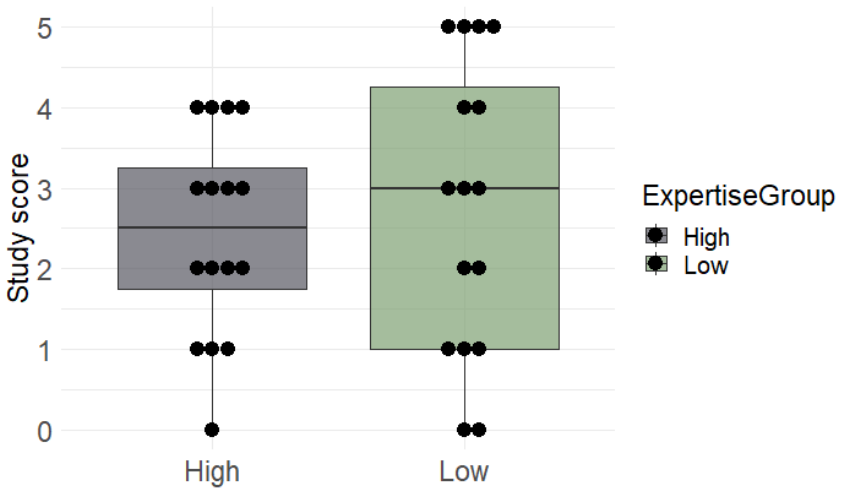


Figure 4.12: Boxplot of the study scores for the two expertise groups; high and low.

A Breusch-Pagan test showed signs of heteroscedasticity ($BP(1) = 4.0055$, $P = .04535$). Therefore, a two-sample independent Student's t-test, which assumes equal variances, could not be performed. Instead, a two-sample independent Welch's t-test with significance level $\alpha = .05$ was performed. The test showed no significant difference in study scores between the low expertise group ($M = 2.7500$, $SD = 1.807392$) and the high expertise group ($M = 2.4375$, $SD = 1.263263$) conditions; ($t(26.832) = -0.56687$, $p = .5755$). Further tests and results can be seen in subsection F.1.1 in Appendix F.

Conditions within expertise groups

The study scores between the conditions within the expertise groups were also analyzed.

Figure 4.13 shows boxplots for the study scores for each of the conditions within each expertise group. Figure 4.13a show the study scores for the low group and Figure 4.13b shows the scores for the high group. Please note that the sample size of the divided expertise groups is small, and the results presented here should therefore be taken with a grain of salt.

A Shapiro-Wilk normality test did not show signs of non-normality in study scores for the control group within the low expertise group ($W = 0.8965$, $p = .2324$), nor for the help group ($W = 0.82779$, $p = .07624$). Furthermore, there was

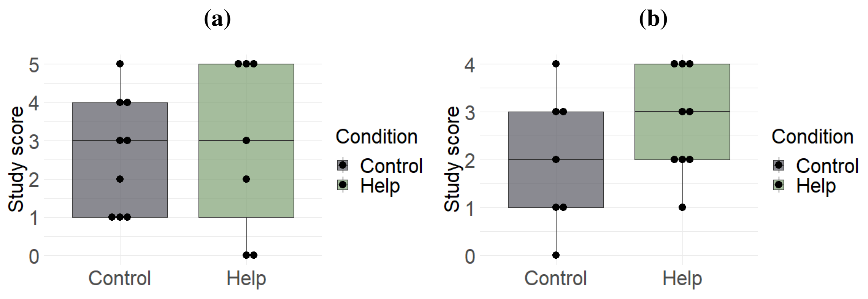


Figure 4.13: Study scores for the two conditions, control and help, within each expertise group. **a)** shows study scores for the low expertise group. **b)** shows study scores for the high expertise group

no sign of non-normality for the control group within the high expertise group ($W = 0.95244$, $p = .7518$), nor the help group ($W = 0.88376$, $p = .1719$). The Breusch-Pagan tests showed signs of non-heteroscedasticity both for the low expertise group ($BP(1) = 3.4076$, $p = .0649$) and the high group ($BP(1) = 1.0841$, $p = .2978$). Furthermore, a two-sample independent Student's t-tests with significance level $\alpha = .05$ did not show any significant difference in study scores between the low expertise control group ($M = 2.667$, $SD = 1.5$) and the low expertise help group ($M = 2.857$, $SD = 2.268$) conditions; ($t(14) = -0.20233$, $p = 0.8426$), nor between the high expertise control group ($M = 2$, $SD = 1.414$) and high expertise help group ($M = 2.778$, $SD = 1.093$) conditions; ($t(14) = -1.2438$, $p = 0.234$). Further tests and results can be seen in subsection F.1.1 in Appendix F.

4.2.2 Number of fixations on lines with bugs

Figure 4.14 shows a boxplot of total number of fixations on all bugs for the two expertise groups.

A Shapiro-Wilk test was run to test for normality in total number of fixations on all bugs for the expertise groups. The test show no sign of non-normality for the low expertise group ($W = 0.97832$, $p = .9488$), nor for the high expertise group ($W = 0.9003$, $p = .08128$). Furthermore, a Breusch-Pagan test yielded a result of non-heteroscedacity ($BP(1) = 1.0426$, $p = .3072$). Therefore, a two-sample independent Student's t-test with significance level $\alpha = .05$ was performed, which showed no significant difference in total number of fixations between the low expertise group ($M = 226.0625$, $SD = 124.6507$), and the high expertise group

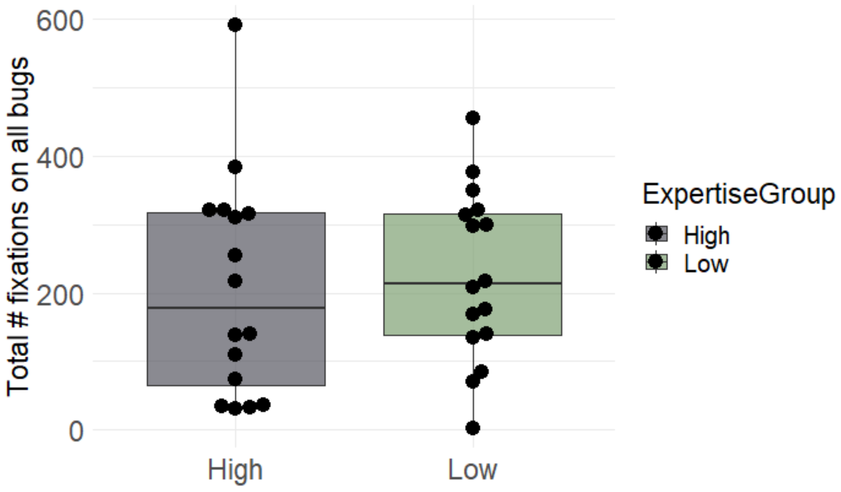


Figure 4.14: Boxplot of total number of fixations on all bugs for the two expertise groups; high and low.

(M = 206.9375, SD = 160.6231), conditions; ($t(30) = -0.37626$, $p = .7094$). Tests and results for all tasks can be seen in subsection F.1.2 in Appendix F.

4.2.3 Time spent on lines with bugs

Figure 4.15 shows a boxplot of total normalized fixation duration on all bugs for the two expertise groups.

A Kruskal-Wallis rank sum test was performed to test for difference in total normalized fixation duration on bugs for the two expertise groups. The test showed no significant difference between the low expertise group (M = 0.0992, SD = 0.0840) and the high expertise group (M = 0.1006, SD = 0.0667), conditions; ($\chi^2 = 0.035511$, $df = 1$, $p = .8505$)

4.2.4 Distance to lines with bugs

Figure 4.16 shows two diagrams representing number of fixations in different percentiles away from the bugs on the tasks. Figure 4.16a shows the total number of the different percentiles across all tasks and bugs, and Figure 4.16b shows a boxplot for the 75 percentile on task five for the two expertise groups.

Tests and results for all percentiles and all tasks can be seen in subsection F.1.4

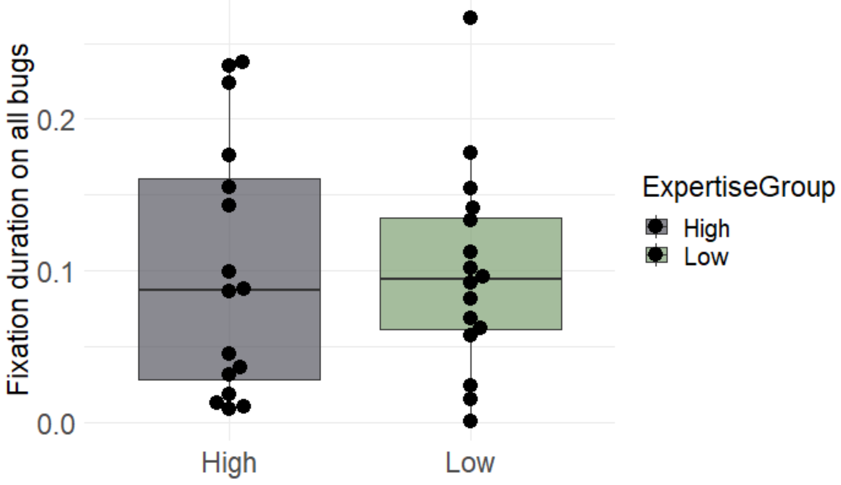


Figure 4.15: Boxplot of total normalized fixation duration on all bugs for the two expertise groups; high and low.

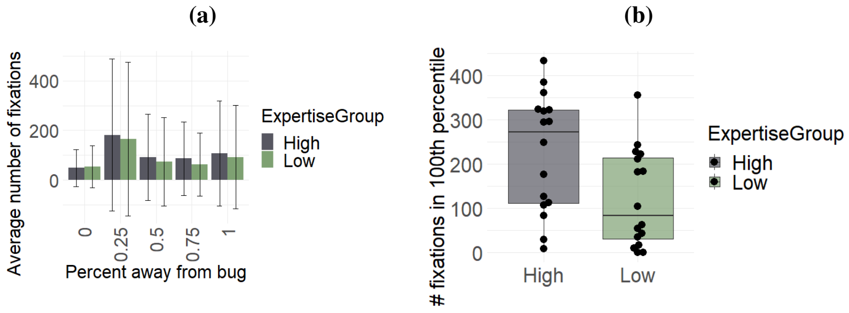


Figure 4.16: Percent distance from the bugs for the expertise groups; high and low. **a)** shows a histogram of mean and standard deviation for number of fixations in the different percentiles from the bug. **b)** shows a boxplot of number of fixations in the 75 percentile on task five for the two expertise groups.

in Appendix F. A Shapiro-Wilk test was performed to test for normality in total number of fixations in the 75 percentile away from the bug on task five for the two expertise groups. The test did not show signs of non-normality for the low expertise group ($W = 0.89339$, $p = 0.06304$), nor for the high expertise group ($W = 0.93208$, $p = 0.2629$). A Breusch-Pagan test of heteroscedacity between the two groups yielded a result of non-heteroscedacity ($BP(1) = 1.3428$, $p = 0.2465$). Finally, a two-sample independent Student's t-test with significance level $\alpha = .05$ showed a significant difference in total number of fixations in the 75 percentile away from the bug on task five for the low expertise group ($M = 122.1875$, $SD = 110.0007$) and the high expertise group ($M = 227.0625$, $SD = 134.0176$), conditions; ($t(30) = 2.4195$, $P = 0.02181$), meaning the high expertise group looked more on the lines 75 percent from the bugs on task five than the low expertise group.

4.2.5 Time spent on tasks

Figure 4.17 shows the average time in seconds the high and low expertise groups and the expert spent per task.

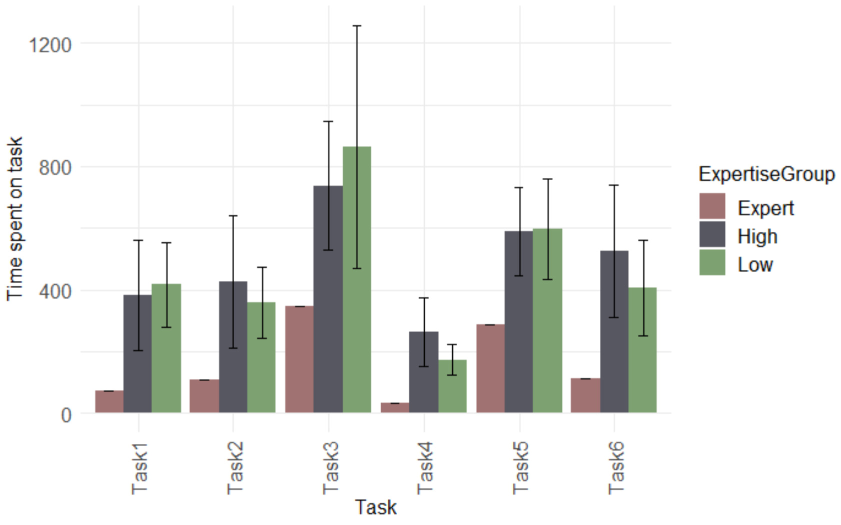


Figure 4.17: Average time spent per task in seconds for the two expertise groups, high and low, as well as the expert. Error bars represent the standard deviation for each task and group.

A Kruskal-Wallis rank sum test was performed to test for difference in average time spent per task between the high and low expertise groups. The test did not show a significant difference between the low expertise group ($M = 396.8658$, $SD = 252.0357$) and the high expertise group ($M = 396.2192$, $SD = 235.7252$) conditions; ($\chi^2 = 0.0046327$, $df = 1$, $p = .9457$). Further tests and results for individual tasks can be seen in subsection F.1.3 in Appendix F.

4.3 Performance groups

This section will present analysis results for the performance groups.

4.3.1 Study scores

Help groups within performance groups

In addition to being split into expertise groups, all participants were split into high and low performance groups based on their scores on the study tasks. The difference in study scores between the two groups is not that interesting, seeing as the participants were split based on the scores. Therefore, there is a difference in study scores between the two groups. Instead, this section will focus on the difference in study scores between conditions within each of the performance groups. Again, when splitting the two performance groups further into conditions, the sample sizes are small, and the results should be taken carefully.

Figure 4.18 displays the study scores of each condition within each performance group as boxplots. Figure 4.18a shows the study scores for the two conditions within the low performance group, and Figure 4.18b for the high performing group.

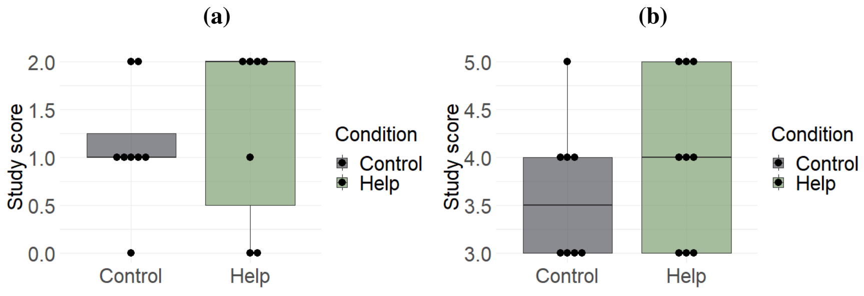


Figure 4.18: Study scores for the conditions, control and help, within each performance group. **a)** shows study scores for the low performance group. **b)** shows study scores for the high performance group

To test for significant difference within the two groups, Kruskal-Wallis rank sum tests were performed. The tests showed no significant difference between the low performing control group ($M = 1.125$, $SD = 0.641$) and the low performing help group ($M = 1.286$, $SD = 0.951$) conditions; ($\chi^2 = 0.3125$, $df = 1$, $p = .5762$), nor between the high performing control group ($M = 3.625$, $SD = 0.744$) and the high performing help group ($M = 4$, $SD = 0.866$) conditions; ($\chi^2 = 0.85594$, $df = 1$, $p = .3549$). Further tests and results can be seen in subsection F.1.1 in Appendix F.

4.3.2 Number of fixations on lines with bugs

Figure 4.19 shows a boxplot of total number of fixations on all bugs for the two performance groups.

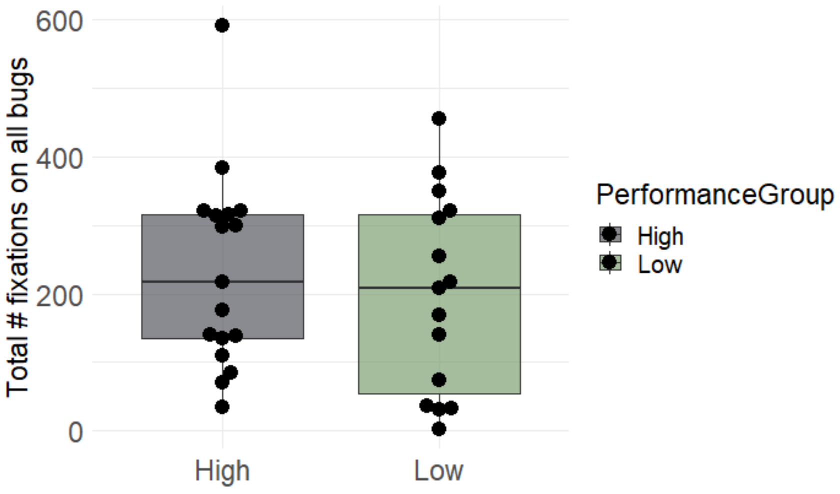


Figure 4.19: Boxplot of total number of fixations on all bugs for the two performance groups; high and low.

A Shapiro-Wilk normality test was performed to test for normality in total number of fixations on all bugs for the low and high performance groups. The test showed no sign of non-normality between the low performance group ($W = 0.94134$, $p = .3995$), nor the high performance group ($W = 0.91904$, $p = .1422$). Furthermore, a Breusch-Pagan test yielded a result of non-heteroscedacity ($BP(1) = 0.0036977$, $p = .9515$). Therefore, a two-sample independent Student's t-test with significance level $\alpha = .05$ was performed, which showed no significant difference in total number of fixations on the bugs between the low performance group ($M = 226.0625$, $SD = 124.6507$) and the high performance group ($M = 206.9375$, $SD = 160.6231$), conditions; ($t(30) = 0.66488$, $p = .5112$). Tests and results for all tasks can be seen in subsection F.1.2 in Appendix F.

4.3.3 Time spent on lines with bugs

Figure 4.20 shows a boxplot of total normalized fixation duration on all bugs for the two performance groups.

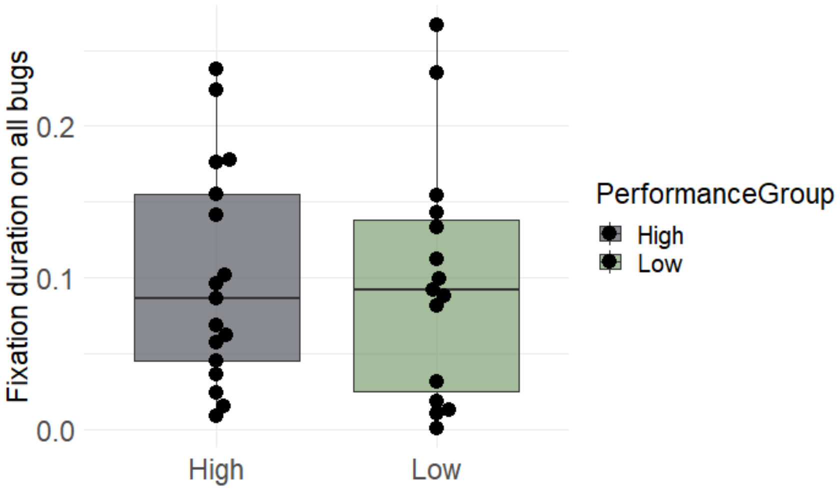


Figure 4.20: Boxplot of total normalized fixation duration on all bugs for the two performance groups; high and low.

A Shapiro-Wilk test of normality was performed to test for normality in total normalized fixation duration on the bugs. The test showed no sign of non-normality for the low performing group ($W = 0.9199$, $p = .192$), nor for the high performing group ($W = 0.92955$, $p = .2139$). A Breusch-Pagan test yielded a result of non-heteroscedacity between the two groups ($BP(1) = 0.20296$, $p = .6523$). Furthermore, a two-sample independent Student's t-test with significance level $\alpha = .05$ showed no significant difference between the low performing group ($M = 0.0987$, $SD = 0.0798$) and the high performing group ($M = 0.1009$, $SD = 0.0722$) with regards to total normalized fixation duration on the bugs, conditions; ($t(30) = 0.080217$, $p = .9366$). Tests and results for individual tasks can be seen in subsection F.1.3 in Appendix F.

4.3.4 Distance to lines with bugs

No significant difference was found regarding distance to lines with bugs for the two performance groups. Tests and results can be seen in subsection F.1.4 in Appendix F.

4.3.5 Time spent on tasks

Figure 4.21 shows the average time in seconds the high and low performance groups and the expert spent per task.

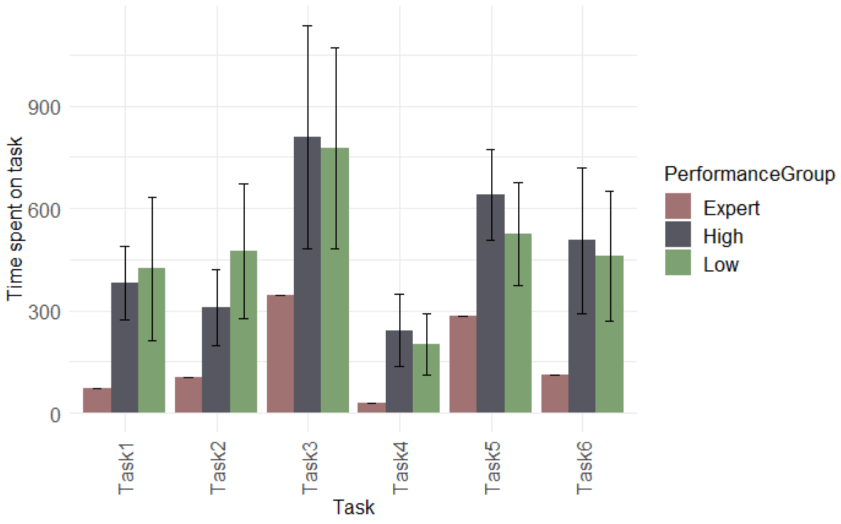


Figure 4.21: Average time spent per task for the two performance groups, high and low, and the expert. Error bars represent the standard deviation for each task and group.

A Kruskal-Wallis rank sum test was performed to test for a difference in average time spent per task between the high and low performance groups. The test showed a significant difference between the low performance group ($M = 363.511$, $SD = 238.2681$) and the high performance group ($M = 425.361$, $SD = 245.1947$) conditions; ($\chi^2 = 4.5843$, $df = 1$, $P = .03227$), meaning the high performance group spent, on average, more time per task than the low performing group.

Chapter 5

Discussion

This thesis aimed to investigate the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novice programmers how to comprehend and debug code. A study exposing novice programmers to an expert's eye-gaze during debugging and code comprehension sessions has been conducted to achieve this goal.

The results show no significant difference in study scores between the two conditions or between the expertise groups. Furthermore, it shows no significant difference in number of fixations or fixation duration on lines containing bugs between the conditions, expertise groups, or performance groups. However, the results show a significant difference in fixation distance to bugs between the conditions on task one and between the expertise groups on task five. In addition, a slight statistically insignificant negative correlation between study score and percent of fixations that was on a line with a bug after consulting the help section has been shown. Moreover, the help group spent significantly more time per task than the control group and that the high performance group spent more time per task than the low performance group.

5.1 Results and their implications

Before starting the study, each participant was tasked with a pretest in order to investigate and confirm that they were novices and of the same level of expertise. The analysis of the pretest scores showed no significant difference in the number of correct scores between the conditions. This means that, even though some participants did get a relatively high pretest score of 8 out of 10, the study had a surplus of novices. This is important as it reduces the bias of the result towards expertise and because the goal of this thesis was concentrated around novice programmers.

Secondly, the study scores of the participants were analyzed. Before the analysis, each participant's answers to the tasks were manually graded as they provided textual answers. As previously mentioned, this grading was done anonymously, meaning the grader did not know if the participant was in the control or help group. The analysis of the study scores showed no significant difference between the help group and the control group, indicating that exposure to the expert's eye-gaze might not impact novices' debugging and comprehension abilities. The answers were graded strictly binary; each answer was either correct or incorrect. When grading, however, the grader observed many answers that were close to correct, especially on tasks two and six. Many of the participants had correctly identified the values being printed in task two but had the wrong output format. E.g., line breaks where no line break should be present or missing a period. In addition, some participants provided the answer 35.5 to task six, when the correct answer was supposed to be 35.0. The removal of the decimal .5 was done in one out of seven methods in the task code that contributed to the final output, meaning some participants did understand what a majority of the code did. The outcome of the analysis of the study scores might have been different if a more relaxed grading had been adopted by, for instance, giving partial points to partially correct answers. Including partially correct answers could help indicate guidance toward the correct answer by the expert's eye-gaze.

Crosby and Stelovsky [3] found that highly experienced programmers tend to spend more time concentrating on meaningful areas in the code than the low experienced programmers, which is inconsistent with the findings in this thesis. The analysis of the number of fixations on meaningful areas showed no significant difference between the high and low expertise participants. What is important to note, however, is that the highly experienced participants in Crosby and Stelovsky [3]'s study were college graduates and Ph.D. faculty members, and the low experienced participants were 2$^{\text{nd}}$ semester students. In contrast, all the participants in this thesis' study were undergraduate students divided into expertise groups based on their pretest scores. Therefore, this thesis' finding of no significant difference in number of fixations on meaningful areas between the expertise groups is not unexpected when compared to the expertise split conducted by Crosby and Stelovsky [3].

The analysis conducted in this thesis did not show a significant difference in number of fixations on meaningful areas between the help and control groups, nor between the high and low performance groups. However, Crosby and Stelovsky [3] found that highly experience programmers spent more time concentrating on meaningful areas. This could further infer that time spent on meaningful areas could indicate a participant's expertise. Furthermore, assuming that programmers with a higher experience overall have higher performance, there should be a connection between the number of meaningful fixations and performance. Since the study conducted for this thesis did not find a significant difference in the number of fixations on meaningful areas between the expertise groups, this indicates that the exposure of novices to an expert's eye-gaze does not provide higher performance.

However, the analysis showed more fixations on the second bug in the fifth task for the help group compared to the control group. As this bug was extremely subtle, even the expert had some trouble identifying it and had to be guided towards identification; one could argue that the help section might have provided the help group with higher performance than the control group. This is further supported by the findings of Jessup et al. [5] that experts have a higher fixation count than novices and Turner et al. [21] that experts have a higher fixation rate than novices on lines containing bugs. Moreover, it is supported by Yoon and Narayanan [2] that found that higher accuracy in a problem can be achieved by paying attention to critical components.

This thesis's analysis of fixation duration on lines containing bugs showed no significant difference between the two conditions, expertise groups or performance groups. This is expected as no difference in the number of fixations on lines containing bugs was shown. Interestingly, though, is the result of no significant difference in fixation duration on the second bug of task five for the conditions, as there was a significant difference in the number of fixations on this specific bug. One would expect the total fixation duration to be higher as the total number of fixations was higher. Assuming the help section did provide an expertise advantage to the help group, this finding is supported by Turner et al. [21] that found no difference in fixation duration on lines containing bugs between experts and novices and does not necessarily mean that the expert's eye-gaze did not provide help in identifying bugs.

Another measurement to test the usefulness of the expert's eye-gaze was distances to the lines with bugs after looking at the help. The difference in number of fixations in different percentile distances from the bug after looking at the help could be an indicator of the effect of the exposure to the expert's eye-gaze. This thesis shows a significant difference between the conditions and the expertise groups on specific tasks, but not between the performance groups.

The analysis showed that the control group looked more at lines furthest away, in the 100 percentile, from the bug on the first task than the help group. For the first task at least, this shows that the help group concentrated more on meaningful areas, as the most meaningful areas of the first task were centered around the bug. The task did have some interesting code in the lines furthest away from the bug. However, this was code that was meant to be read only once. It is possible that the help group was guided toward the more meaningful areas by the expert's eye-gaze since the expert only had to read the lines furthest away from the bug once. However, there was no difference between the two conditions for the other percentiles. This indicates that the help section may not have contributed to the lower number of fixations on the lines furthest away from the bug.

Furthermore, the analysis showed a significant difference in number of fixations in the 75 percentile away from the bug on task five between the expertise groups. Surprisingly though, is the fact that it was the low expertise group that had the lowest number of fixations in this percentile. This is inconsistent with the findings of Crosby and Stelovsky [3] that high expertise programmers spend more time concentrating on meaningful areas than low expertise. One reason for this deviation, may be that the high expertise group focused more on comprehending the entire code to find the root of the bug and put it in context of the rest of the code. Put in context with Jessup et al. [5]'s finding that high expertise programmers have a higher fixation count than novices, one could argue that this is true for the high expertise novice programmers in this study as well. However, as this finding is only visible on one of the tasks and one of the percentiles, one cannot say anything for sure about the difference between high and low expertise novices with regards to percentage distance from lines with bugs.

For the help group, a slight, statistically insignificant, negative correlation between the study scores and the percentage of first fixations after consulting the help section that was on a line with a bug was found. The correlation test presented a p-value that showed no significant correlation. However, this might be due to the low sample size in the help group ($N = 16$). A negative correlation of -0.4146 is generally considered a medium correlation. Nevertheless, it is a surprising finding, as one would expect the study scores to increase as the percentage of first fixations on bugs increases.

On the other hand, the definition of a problem being in a student's zone of proximal development (see Figure 2.2 in section 2.2) is that the student should be able to solve the problem with guidance. The results of this analysis, however, show that after getting help, the participants correctly identified the meaningful areas but did, for the most part, not manage to answer the problem correctly. This indicates that the tasks may either have been in the far upper end of the participants' zone of proximal development or the right hand side of Figure 2.2.

The analysis of time spent per task showed that the help group spent on average more time than the control group per task. This finding is inconsistent with Stein and Brennan [7], who found that the participants exposed to another person's eye-gaze found bugs faster than those not exposed. An important distinction between the study in this thesis and that of Stein and Brennan [7], is that the participants in this thesis were novices and not professional programmers. Another important distinction is that the participants in Stein and Brennan [7] first watched the eye-gaze of a person solving the task and then identified the bugs. In contrast, the participants in this thesis' study had the option to interactively choose when to watch the expert's eye-gaze during the tasks.

In this thesis, the extra time participants spent on the help section was discarded before analyzing whether there was a difference in time spent per task between the two conditions. It is possible that the help group spent more time than the control group exploring and covering all the code that the expert viewed and then did their own exploring afterward, being afraid to miss something the expert had looked at. The control group did not have this option and was only focused on reading the code to find the bug or comprehend the code.

Furthermore, when analyzing the average time spent per task, there was no significant difference between the high and low expertise groups, which further strengthens the assumption that all participants were novices at approximately the same level, as found by analyzing the pretest scores.

However, the high performing group spent, on average, more time per task than the low performing group. Seeing as Najar et al. [4] found that low performing participants did not use the provided help, and Sharma et al. [19] found that they were more prone to boredom than the high performers, one can draw a line between the help group and high performing group, and the control group and the low performing group. The low performing group might have experienced more boredom than the high performing group, making them rush through the tasks and ignore the help as found by Najar et al. [4]. On the other hand, the high performing group might have been less bored and motivated by the help section. This indicates that exposure to an expert's eye-gaze could motivate novices to finish a task correctly.

5.2 Limitations

It should be noted that the study progressed over seven weeks. Seven weeks in itself is not a long time but put in a university and semester time perspective; a lot can be taught. This means that the later participants had attended more lectures than the early participants and had therefore learned more Java programming and concepts. This might have impacted the study scores. However, as seen by the analysis of the pretest scores, all participants were at the same level of expertise. Furthermore, as indicated by eye-tracking metrics, Yenigalla et al. [30] found no significant change in novices' learning throughout two introductory courses in programming.

Another limitation of this study is that halfway through, the environment in which the participants completed the study changed. The first half completed the study in the UX-lab at NTNU, whereas the second half completed it in the supervisor's office. Those who completed the study in the lab may have felt more relaxed as it was a more relaxing environment than an office.

Furthermore, the gaze samples of the participants were low. This means that the eye-gaze analysis of number of fixations, duration, and distance to bugs may not be accurate, as the actual number and duration of fixations would have been higher with more gaze samples. In addition, the accuracy of the collected gaze samples was not the best, with a mean accuracy of 53 pixels. The AOIs used in the analysis, for the most part, were single code lines; an accuracy of 53 pixels could have contributed to categorizing an AOI hit falsely.

5.3 Future work

No significant results in this thesis clearly show that exposing novices to an expert's eye-gaze either helps the novice, hampers their abilities, or has no impact at all. Some signs might suggest that the expert's eye-gaze does, in fact, guide the novices toward the correct place in the code but not toward the correct solution to the problem. As stated in section 5.1, it might be because of the task difficulty. Future work should therefore include further investigation on the effect of exposing novice programmers to an expert's eye-gaze, with problems being more well-placed in the participant's zone of proximal development.

Furthermore, future work should incorporate the other help types, ”expert choose” and ”always-on,” as first planned for the study in this thesis. This should be done to investigate different forms of visualizing the expert's eye-gaze, as the ”on-demand”-type help included in this thesis' study might not have been the best way to provide the participants help.

In addition, other ways of grading the answers to the tasks should be explored. This could be, for instance, to give partially correct answers a score other than 0, as was done in this thesis. A more relaxed grading could give other results than a strict binary grading. Including partially correct answers could, in fact, help indicate guidance toward the correct answer by the expert's eye-gaze.

Moreover, future work should consider this thesis' limitations by maximizing the accuracy of the eye-tracker, executing the study in the same place for all participants, and limiting the time gap between participants.

This thesis has focused on the debugging part of the study. This was done mainly because one cannot debug without comprehending the code; therefore, code comprehension was also implicitly analyzed. Future work, however, should be more focused on code comprehension regarding visualizing an expert programmer's eye-gaze to novices. Moreover, interesting analysis for future work includes analysis of code depth, eye movement speeds, and number of scanpaths.

Once the groundwork of visualizing an expert programmer's eye-gaze to novices is fully complete, future work should research the effects of implementing it in computing education. This could, for instance, be as a supplement to students' assignments or the lecturer using an eye-tracker while giving a lecture.

Chapter 6

Conclusion

This thesis has presented literature on how eye-tracking is used in programming and the differences between experts and novices when programming. Furthermore, it has presented the planning, preparation, execution, analysis, and results of a study exposing novice programmers to an expert's eye-gaze during debugging and code comprehension. This has been done by implicitly answering the research questions in section 1.2. Here, the research questions will be answered explicitly.

The literature study answered the first and second research questions in section 2.3. **RQ1** asked how eye-tracking is used in programming. It was shown that the interest and acceptance in using eye-trackers in programming are increasing, with the increasing number of published papers on the topic. Furthermore, it was shown that eye-trackers are used to identify reading behaviors, strategies, and gaze patterns in programming. **RQ2** asked what the differences are between experts and novices when programming. Through related work, this thesis showed considerable differences between experts and novices when programming; Highly experienced programmers tend to read more source code elements and meaningful areas than low experience programmers, and low experience programmers tend to focus more on comments and comparisons. Furthermore, advanced programmers seem to be using help to supplement a problem more frequently and differently than novices. Moreover, novices tend to have a lower fixation rate on lines with bugs than experts. Related work also showed little to no difference in fixation duration between experts and novices when programming.

RQ3 considered what effect exposing novice programmers to an expert programmer's eye-gaze has on their comprehension and debugging abilities, which was answered through the study performed in this thesis. First, a system used in the study to present both an expert and novices with code snippets was created. In addition, the system contained a pretest for the novices to test their level of expertise. Furthermore, the system incorporated a help section for half of the novices where the expert's eye-gaze was visualized.

Then, an expert's eye-gaze was recorded while they completed a set of code problems. Later, 32 novices were evenly split into a control group and a help group. Both groups did a pretest to test their level of expertise and completed the same set of problems as the expert. The help group had the option to view the expert's eye-gaze and code snippet side-by-side. Once the data was collected, it was analyzed to answer **RQ3**. This was done by finding differences between the control and help groups and splitting the participants into high and low expertise and performance groups to investigate differences further.

The analysis showed no difference in study scores between the conditions or the expertise and performance groups. Furthermore, it showed no difference in number of fixations or fixation duration on meaningful areas in the code between the different groups. However, there was a significant difference regarding distances to meaningful areas of the code. The control group fixated more on lines furthest away from the meaningful areas on task one. Moreover, the high expertise group fixated more on the 75 percentile distance from the bug on task five than the low expertise group. Furthermore, a small, statistically insignificant, negative correlation between the score on the study tasks and the number of first fixations after consulting the help section that was on a line with a bug was found. The analysis also showed that, even without the time spent consulting the help section, the help group spent on average more time per task than the control group. Lastly, the analysis showed that high performers spent more time per task than low performers.

This thesis aimed to investigate the effect of visualizing an expert programmer's cognitive code comprehension and debugging process through eye-tracking as a basis for teaching novice programmers how to comprehend and debug code. The goal of the thesis has been achieved by answering the research questions. This thesis has presented a way of visualizing an expert's eye-gaze to novices completing code problems and has laid the groundwork for research on how this can be implemented in computing education.

Bibliography

[1] Merriam-Webster. Problem Solving. `https://www.merriam-webster.com/dictionary/problem-solving`, 2021. Online; accessed October 12, 2021.

[2] Daesub Yoon and N. Hari Narayanan. Mental imagery in problem solving: An eye tracking study. In *Proceedings of the 2004 symposium on Eye tracking research & applications*, ETRA '04, pages 77–84. Association for Computing Machinery, 2004. doi:10.1145/968363.968382.

[3] Martha E. Crosby and Jan Stelovsky. How do we read algorithms? a case study. *Computer*, 23(1):25–35, 1990. doi:10.1109/2.48797.

[4] Amir Shareghi Najar, Antonija Mitrovic, and Kourosh Neshatian. *Utilizing Eye Tracking to Improve Learning from Examples*, pages 410–418. Springer, Cham, 2014. doi:10.1007/978-3-319-07440-5˙38.

[5] Sarah Jessup, Sasha M. Willis, Gene Alarcon, and Michael Lee. Using eye-tracking data to compare differences in code comprehension and code perceptions between expert and novice programmers. In *Proceedings of the 54th Hawaii International Conference on System Sciences*, pages 114–123. Hawaii International Conference on System Sciences, 2021. doi:10.24251/hicss.2021.013.

[6] Salwa Aljehane, Bonita Sharif, and Jonathan 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*,

ETRA '21, pages 1–6. Association for Computing Machinery, 2021. doi:10.1145/3448018.3457424.

[7] Randy Stein and Susan E. Brennan. Another person's eye gaze as a cue in solving programming problems. In *Proceedings of the 6th International Conference on Multimodal Interfaces*, ICMI '04, pages 9–15. Association for Computing Machinery, 2004. doi:10.1145/1027933.1027936.

[8] Sander B. Lindberg. How eye tracking can be used in problem solving. Trondheim: Norwegian University of Science and Technology, 2021.

[9] Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, 67:79–107, 2015. doi:10.1016/j.infsof.2015.06.008.

[10] Florian Hauser, Jürgen Mottok, and Hans Gruber. Eye tracking metrics in software engineering. In *Proceedings of the 3rd European Conference of Software Engineering Education*, ECSEE'18, pages 39–44. Association for Computing Machinery, June 2018. doi:10.1145/3209087.3209092.

[11] Lev Semenovich Vygotsky and Michael Cole. *Mind in society: Development of higher psychological processes*. Harvard university press, 1978.

[12] Joakim Caspersen, Thomas De Lange, Tine S. Prøitz, Tone D. Solbrekke, and Bjørn Stensaker. Learning about quality - perspectives on learning outcomes and their operationalisations and measurement. *Oslo: Department of Educational Research, University of Oslo*, 1:10, 2011.

[13] Peter E. Doolittle. Understanding cooperative learning through vygotsky's zone of proximal development. In *Lilly National Conference on Excellence in College Teaching*. ERIC, 1995.

[14] Barbara Kitchenham. Procedures for performing systematic reviews. *Keele, UK, Keele University*, 33(2004):1–26, 08 2004.

[15] Teresa Busjahn, Carsten Schulte, Bonita Sharif, Andrew Begel, Michael Hansen, Roman Bednarik, Paul Orlov, Petri Ihantola, Galina Shchekotova, and Maria Antropova. Eye tracking in computing education. In *Proceedings of the tenth annual conference on International computing education research*, ICER '14, pages 3–10. Association for Computing Machinery, 2014. doi:10.1145/2632320.2632344.

[16] Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. Tracing software developers' eyes and interactions for change tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 202–213. Association for Computing Machinery, 2015. doi:10.1145/2786805.2786864.

[17] Tamara van Gog, Halszka Jarodzka, Katharina Scheiter, Peter Gerjets, and Fred Paas. Attention guidance during example study via the model's eye movements. *Computers in Human Behavior*, 25(3):785–791, 2009. doi:https://doi.org/10.1016/j.chb.2009.02.007.

[18] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. Measuring and modeling programming experience. *Empirical Software Engineering*, 19(5):1299–1334, 2014. doi:10.1007/s10664-013-9286-4.

[19] Kshitij Sharma, Sofia Papavlasopoulou, and Michail Giannakos. Faces don't lie: Analysis of children's facial expressions during collaborative coding. In *FabLearn Europe / MakeEd 2021 - An International Conference on Computing, Design and Making in Education*, FabLearn Europe / MakeEd 2021. Association for Computing Machinery, 2021. doi:10.1145/3466725.3466757.

[20] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. Itrace: Enabling eye tracking on software artifacts within the ide to support software engineering tasks. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, pages 954–957. Association for Computing Machinery, 2015. doi:10.1145/2786805.2803188.

[21] Rachel Turner, Michael Falcone, Bonita Sharif, and Alina 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*, ETRA '14, pages 231–234. Association for Computing Machinery, 2014. doi:10.1145/2578153.2578218.

[22] Sander B. Lindberg. Master-study-system. `https://github.com/skanin/master-study-system`, 2022. commit-hash: 5143738217e5ab425e7cbde4056e510374757bb8.

[23] Sander B. Lindberg. Master-study-system-backend. `https://github.com/skanin/master-study-system-backend`, 2022. commit-hash: a0a9f9f5647e513d8fa3dd415f07d8a08fb53ebf.

[24] P.B. Kruchten. The 4+1 view model of architecture. *IEEE Software*, 12(6): 42–50, 1995. doi:10.1109/52.469759.

[25] Tobii Pro AB. Tobii pro lab. `http://www.tobiipro.com/`, 2021. Computer software, version 1.181.

[26] Tobii Pro AB. *Pro Lab User Manual.* Tobii Pro AB, 2021. v 1.181, url: `https://www.tobiipro.com/siteassets/tobii-pro/user-manuals/Tobii-Pro-Lab-User-Manual/`. Online; accessed May 20, 2022.

[27] Felipe Pezoa, Juan L Reutter, Fernando Suarez, Martín Ugarte, and Domagoj Vrgoč. Foundations of json schema. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 263–273. International World Wide Web Conferences Steering Committee, 2016. doi:10.1145/2872427.2883029.

[28] Sander B. Lindberg. Master-support-scripts. `https://github.com/skanin/master-support-scripts`, 2022. commit-hash: 3629b89f61f118140f29add36e971eae70dceb6d.

[29] Tobii Pro AB. *Accuracy and precision Test report.* Tobii Pro AB, 2015. url: `https://www.tobiipro.com/siteassets/tobii-pro/accuracy-and-precision-tests/tobii-pro-x3-120-accuracy-and-precision-test-report.pdf`. Accessed May 20, 2022.

[30] Leelakrishna Yenigalla, Vinayak Sinha, Bonita Sharif, and Martha Crosby. How novices read source code in introductory courses on programming: An eye-tracking experiment. In *International Conference on Augmented Cognition*, pages 120–.131. Springer International Publishing, 2016. doi:10.1007/978-3-319-39952-2˙13.

Appendix A

Diagrams

A.1 Logical view

Figure A.1 shows a more extensive version of the logical view diagram than the one in Figure 3.7. It contains methods and variables in the different components of the study system.

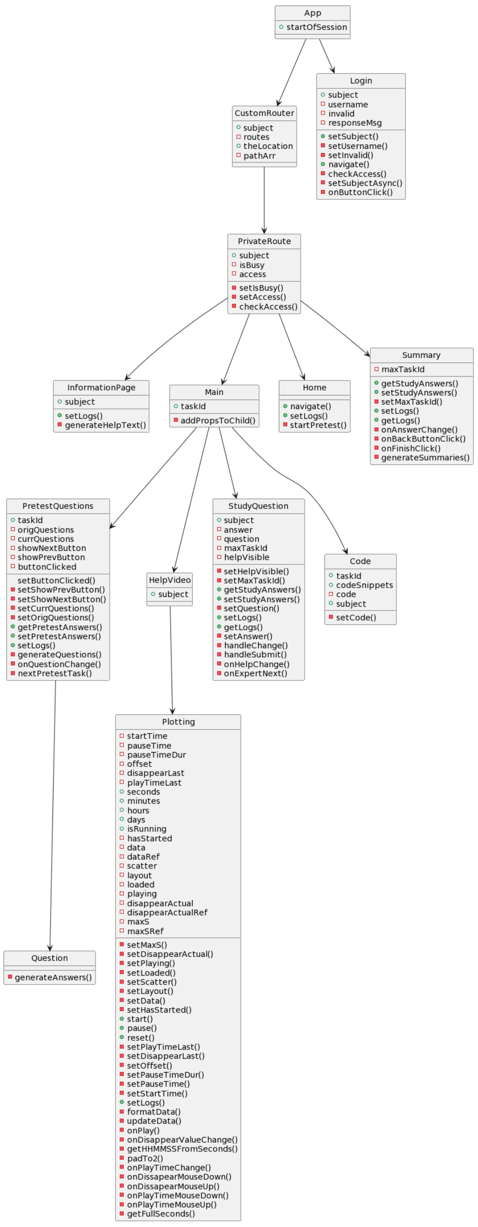


Figure A.1: Logical view of the study system. A more extensive version of the diagram in the thesis. The diagram contains methods and variables for the components.

Appendix B

Information letter

Request for participation in a research project

"Eye-tracking to enhance programming performance."

This is an inquiry about participation in a research project where the primary purpose is to investigate the effect of exposing novice programmers to an expert programmer's eye-gaze. In this letter, we will give you information regarding the purpose of the project and what your participation will involve.

Purpose

The purpose of the study is to investigate the effect of exposing novice programmers to an expert programmer's eye-gaze. Firstly, the expert programmer will complete a series of programming tasks (debugging and comprehension in Java) while their eye-gaze is collected with an eye-tracker. Then, the novices will complete the same tasks while having their eye-gaze collected. In addition, the novices will be exposed to the expert's eye gaze on the code and answer questions regarding the code snippets.

The study is part of a Master's thesis.

The project participants will be students at NTNU Campus Gløshaugen in Trondheim, Norway, volunteering to participate.

Who is responsible for the research project?

The responsible for the project will be Kshitij Sharma, Associate Professor at the Department of Computer Science (IDI) at NTNU, Trondheim, Norway (see general information section).

Why are you being asked to participate?

You are being asked to participate because you are currently taking, or has recently taken the course "TDT4100", meaning you might be a novice in Java programming. Since the purpose of the study is to investigate the effect on novices, you are a perfect candidate.

What does participation in the project imply?

For the research project's purpose, eye-tracking data will be collected through a Tobii TX120 eye-tracker, and the provided Tobii eye-tracking software. Background information about the participants (year of study, programming experience, etc.) will be collected through Nettskjema. The background information will be collected to determine the level of programming expertise. The programming tasks will be performed in a self-developed application. Participants' answers to the programming tasks will also be collected through this application.

The duration of the user-participation will be approximately one hour, where the participants will complete six programming tasks (comprehension and debugging) with associated questions.

Participants can request to see the questions regarding background information and ask for any additional information regarding any other data collection instrument before giving consent. In addition, they can request a copy of all data stored about them, including their eye-gaze and answers to the programming tasks at any point during or after the study. Furthermore, participation is entirely voluntary, and consent can be withdrawn at any time.

What will happen to the information about you?

All personal data will be treated confidentially in accordance with data protection legislation (GDPR) and used only for the purpose specified in this letter. Only the project group (see general information below) will have access to the personal data. The list of the participating students and experts will be stored in NTNU Sharepoint, according to the data processing agreement between NTNU and Microsoft. Only the researchers and data controllers will have access to the data. Eye-tracking data and answers to the tasks will be stored in computers at NTNU premises and the researchers' personal computers.

The participants will not be recognizable in the publication. The project is scheduled for completion by June 2022. The personal data will be stored for one year after the project's completion – until July 2023 -, so the project leader can continue analysis after the master student's graduation. The project leader will have responsibility for the personal data during this period.

Voluntary participation

It is entirely voluntary to participate in the study. Participants can, at any time, choose to withdraw their consent without stating any reason. If a participant decides to withdraw their consent, all personal data will be deleted.

Participants' rights

Participants have the right to request access to/deletion/limitation/correction of personal data, the right to data portability, and the right to send a complaint to the Data Protection Officer at NTNU or The Norwegian Data Protection Authority about the processing of personal data at any time.

What gives us the right to process your personal data?

We will process your personal data based on your consent.

General information-project group

The leader of the project is Kshitij Sharma, Associate Professor at the Department of Computer Science (IDI) at NTNU, e-mail: XXXXX.XXXXX@ntnu.no (Redacted for appendix in master thesis), address: Sem Sælands vei 9, IT-bygget * 147, phone number: +47 XXX XX XXX (redacted for appendix in master thesis).

If you would like to participate or have any questions concerning the project, please contact:

Sander Bjerklund Lindberg, e-mail: sanderbl@stud.ntnu.no, phone number: +47 XXX XX XXX (redacted for appendix in master thesis), Master student at the Department of Computer Science (IDI) at NTNU.

Data Protection Officer (Personvernombud) at NTNU (Thomas Helgesen, XXXXX.XXXXX@ntnu.no (Redacted for appendix in master thesis))

The study has been notified to the NSD – The Norwegian Centre for Research Data AS (personverntjenester@nsd.no, XX XX XX XX (redacted for appendix in master thesis), and they assessed that the processing of personal data in this project is in accordance with data protection legislation.

Consent for participation in the study

I have received information about the project, and I am willing to give my consent for my participation.

Participant's name: ____

____

(Signed by participant, date)

Appendix C

Pretest tasks

C.1 Pretest task 1

Related question: Which of the following would the Java coding snippet return as its output?.
Answer options: 0, 10, 1, Compile time error

```java
class Super {
   public int index = 1;
}

class App extends Super {

   public App(int index) {
      index = index;
   }

   public static void main(String args[]) {
      App myApp = new App(10);
      System.out.println(myApp.index);
   }
}
```

Listing C.1: Source code for pretest task 1.

C.2 Pretest task 2

Related question: Which of the following combinations would the Java coding snippet print?.
Answer options: 0 1, 1 0, 0 0, null

```
class TestApp {
   protected int x, y;
}

class Main {
   public static void main(String args[]) {
      TestApp app = new TestApp();
      System.out.println(app.x + " " + app.y);
   }
}
```

Listing C.2: Source code for pretest task 2.

C.3 Pretest task 3

Related question: What would be the outcome of this Java coding snippet?.
Answer options: Welcome, Welcome Welcome, Type mismatch error, Run infinite-times

```
class TestApp {
   public static void main(String args[]) {
      for (int index = 0; 1; index++) {
         System.out.println("Welcome");
         break;
      }
   }
}
```

Listing C.3: Source code for pretest task 3.

C.4 Pretest task 4

Related question: Which of the following would the Java coding snippet return as its output?.
Answer options: Welcome, None, Type mismatch error, Run infinite-times

```java
class TestApp {
   public static void main(String[] args) {
      for (int index = 0; true; index++) {
         System.out.println("Welcome");
         break;
      }
   }
}
```

Listing C.4: Source code for pretest task 4.

C.5 Pretest task 5

Related question: Which of the following values would this Java coding snippet print in result?.
Answer options: 0, 1, 2, Compilation error

```java
class TestApp {
   int i[] = { 0 };

   public static void main(String args[]) {
      int i[] = { 1 };
      alter(i);
      System.out.println(i[0]);
   }

   public static void alter(int i[]) {
      int j[] = { 2 };
      i = j;
   }
}
```

Listing C.5: Source code for pretest task 5.

C.6 Pretest task 6

Related question: What does this Java coding snippet print on execution?.
Answer options: Compilation fails, 3, 2, 99

```java
class TestApp {

   public static void main(String args[]) {
      int[] table = { 1, 2, 3, 4, 5 };
      table[1] = (table[2 * 1] == 2 - args.length) ?
         table[3] : 99;
      System.out.println(table[1]);
   }
}
```

Listing C.6: Source code for pretest task 6.

C.7 Pretest task 7

Related question: Which of the following values would this Java coding snippet yield?.
Answer options: 199, 199.5, 200, Invalid number

```java
class TestApp {
   public static void main(String args[]) {
      String text = "199";
      try {
         text = text.concat(".5");
         double decimal = Double.parseDouble(text);
         text = Double.toString(decimal);
         int status = (int)
            Math.ceil(Double.valueOf(text).doubleValue());
         System.out.println(status);
      } catch (NumberFormatException e) {
         System.out.println("Invalid number");
      }
   }
}
```

Listing C.7: Source code for pretest task 7.

C.8 Pretest task 8

Related question: What would this Java coding snippet return as its output?.
Answer options: An exception occurs while instantiating the A class., It'll print ”This is a class A instance”, The program will print null, Compilation error at line number 13

```
 1  class TestApp {
 2
 3     public static void main(String args[]) {
 4        class A {
 5           public String name;
 6
 7           public A(String a) {
 8              name = a;
 9           }
10        }
11
12        Object obj = new A("This is a class A instance");
13        A a = (A) obj;
14        System.out.println(a.name);
15     }
16  }
```

Listing C.8: Source code for pretest task 8.

C.9 Pretest task 9

Related question: What would the this method yield when called?.
Answer options: If a and b both are true, then the output is ”A && B”, If a is true and b is false, then the output is ”!B”, If a is false and b is true, then the output is ”None”, If a and b both are false, then the output is ”None”

```
public void test(boolean a, boolean b) {
      if (a) {
         System.out.println("A");
      } else if (a && b) {
         System.out.println("A && B");
      } else {
         if (!b) {
            System.out.println("!B");
         } else {
            System.out.println("None");
         }
      }
   }
```

Listing C.9: Source code for pretest task 9.

C.10 Pretest task 10

Related question: What will be the output of this Java coding snippet?.
Answer options: abc, abcd, abcde, abcdef

```
class TestApp {
public static void main(String[] args) {
      String obj = "abcdef";
      int length = obj.length();
      char c[] = new char[length];
      obj.getChars(0, length, c, 0);
      CharArrayReader io_1 = new CharArrayReader(c);
      CharArrayReader io_2 = new CharArrayReader(c, 0, 3);
      int i;
      try {
         while ((i = io_2.read()) != -1) {
            System.out.print((char) i);
         }
      } catch (IOException e) {
         e.printStackTrace();
      }
   }
}
```

Listing C.10: Source code for pretest task 10.

Appendix D

Study tasks

D.1 Task 1 - Debug

Related question: Do we get the expected output when running this code? If not - what is the problem and what line(s) contribute(s) to the problem?
Answer: No, we do not get the expected output. Line 6 should be `arr[i] = arr[j]`

```
1  public class Task1 {
2     public static int[] reverse(int[] arr) { // Takes an
         array as input and returns the reversed array
3        int i = 0, j = arr.length - 1, temp;
4        while (i < j) {
5           temp = arr[i];
6           arr[j] = arr[i];
7           arr[j] = temp;
8           i++;
9           j--;
10       }
11       return arr;
12    }
13
14    public static void main(String args[]) {
15       int[] arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
16       arr = reverse(arr);
```

```
        for (int i: arr) {
            System.out.print(i + " ");
        }
        /* Expected output: 10 9 8 7 6 5 4 3 2 1 */
    }
}
```

Listing D.1: Source code for study task 1

D.2 Task 2 - Comprehension

Related question: What is the output of this code?
Answer: 0 0 0 0 1 1 1 2 2 3 .

```
class Task2 {
    public static void main(String[] args) {
        int[] A = {0, 1, 2, 2, 0, 0, 3, 1, 1, 0};
        int[] B = new int[4];
        for (int i = 0; i < A.length; i++) {
            B[A[i]]++;
        }
        for (int j = 0; j < B.length; j++) {
            for (int k = 0; k < B[j]; k++) {
                System.out.print(j + " ");
            }
        }
        System.out.println(".");
    }
}
```

Listing D.2: Source code for study task 2

D.3 Task 3 - Debug

Related question: Which line(s) contains bugs, and what are the bug(s)?
Answer: Line 12 contains a bug. We forget to add one to count. Also, line 67 contains a bug, where `hasItem` returns `false` if store has item and `true` if not.

```java
class ShoppingCart { // Class for simulating a shopping
    cart at a Store
   private Store store;
   private Map<Item, Integer> items = new HashMap<Item,
      Integer>();

   public ShoppingCart(Store store) {
      this.store = store;
   }

   public void addItem(Item item) { // Adds an item to the
      shopping cart, or increases the amount of an
      existing item
      if (!store.hasItem(item)) throw new
         IllegalArgumentException("Item not in store");

      int count = items.get(item) == null ? 0 :
         items.get(item);
      items.put(item, count);
   }

   public void removeItem(Item item) { // Removes an item
      from the shopping cart, or decreases the amount of
      an existing item
      int count = items.get(item) == null ? 0 :
         items.get(item);
      if (count > 1) items.put(item, count - 1);
      else if (count == 1) items.remove(item);
   }

   public double getTotal() { // Returns the total price
      of all items in the shopping cart
      return items.entrySet().stream()
         .mapToDouble(entry -> entry.getKey().getPrice() *
            entry.getValue())
         .sum();
   }

   public void print() { // Prints the items in the
      shopping cart
      for (Item item : items.keySet())
         System.out.println(item.getName() + ": " +
            items.get(item) + "x" + item.getPrice() +
            item.getCurrency());
      System.out.println("Total: " + getTotal());
    }
}

class Item { // Class for simulating an Item in a Store

    private String name, currency;
    private double price;

    public Item(String name, double price, String currency)
        {
        this.name = name;
        this.price = price;
        this.currency = currency;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public String getCurrency() {
        return currency;
    }

}

class Store { // Class for simulating a Store that
    contains Items one can put in a ShoppingCart
    public List<Item> items = Arrays.asList(
        new Item("Sugar", 20.0, "kr"),
        new Item("Milk", 16.5, "kr"),
        new Item("Bread", 25.0, "kr"),
        new Item("Eggs", 24.0, "kr")
    );

    public boolean hasItem(Item item){ // Checks if a Store
        has an item
        return !items.contains(item);
    }
    public static void main(String[] args) {
        Store store = new Store(); // Init a Store
        ShoppingCart shoppingCart = new ShoppingCart(store);
```

```
            // Init a ShoppingCart
        shoppingCart.addItem(store.items.get(0)); // Add an
            item to the ShoppingCart
        shoppingCart.addItem(store.items.get(1)); // Add an
            item to the ShoppingCart
        shoppingCart.print(); // Print the items in the
            ShoppingCart
        /* Expected output: Milk: 1x16.5kr\nSugar:
            1.20.0kr\nTotal: 36.5 */
    }
}
```

Listing D.3: Source code for study task 3

D.4 Task 4 - Comprehension

Related question: What is the output of this code?
Answer: 30\n20

```
class A {
   private final int a = 10;
   private final int b = 20;

   public int C() {
      return a;
   }

   public int D() {
      return b;
   }
}

class B extends A {
   private int a;
   private int b;

   public B(int a, int b) {
      this.a = a;
      this.b = b;
   }

   @Override
```

```java
    public int C() {
        return a;
    }

    public static void main(String[] args) {
        B b = new B(30, 40);
        System.out.println(b.C());
        System.out.println(b.D());
    }
}
```

Listing D.4: Source code for study task 4

D.5 Task 5 - Debug

Related question: Which line(s) contains bugs, and what are the bug(s)?
Answer: Line 24 contains a bug. We are dividing two integers, which results in an integer. Also, line 20 makes it impossible to add more of the same score.

```java
1  import java.util.ArrayList;
2  import java.util.List;
3
4  class CoffeeReview { // Class for storing scores for a coffee made by a person
5      private List<Integer> scores = new ArrayList<Integer>(); // Stores the scores for this coffee review
6      private List<Person> reviewers = new ArrayList<Person>(); // Reviewers in this coffee review
7      private String name; // Name of the coffee made
8      private Person coffeeMaker; // Person that made the coffee
9
10     public CoffeeReview(String name, Person coffeeMaker) {
11         this.name = name;
12         this.coffeeMaker = coffeeMaker;
13     }
14
15     public void addScore(Person reviewer, int score) { // Adds a score to the coffee in this coffee review
16         if(score < 1 || score > 6) throw new
```

```
            IllegalArgumentException("Score must be between 1 and 6");
        if(reviewer == null) throw new IllegalArgumentException("Reviewer cannot be null");
        if(reviewer == this.coffeeMaker) throw new IllegalArgumentException("Reviewer cannot be the owner");
        if (!reviewers.contains(reviewer)) reviewers.add(reviewer);
        if (!scores.contains(score)) scores.add(score);
    }

    public double getAverageScore() { // Gets the average score in this coffee review
        return scores.stream().reduce(0, Integer::sum) / scores.size();
    }
}

class Person { // A person that can make a coffee or be a reviewer of a coffee
    private List<CoffeeReview> coffees = new ArrayList<CoffeeReview>();
    private String name;

    public Person(String name) {
        this.name = name;
    }

    public CoffeeReview addCoffee(String name) {
        // Adds a coffee review for a coffee. The coffee is a fictive coffe made by this person, and exists only as the name of the coffee review
        CoffeeReview coffeeReview = new CoffeeReview(name, this);
        coffees.add(coffeeReview);
        return coffeeReview;
    }

    public double getAverageScore() { // Gets the average score of all coffees made by this person
        return coffees.stream()
                .mapToDouble(CoffeeReview::getAverageScore)
                .average()
                .orElse(0);
    }

    public void reviewCoffee(CoffeeReview coffeeReview, int
        score) { // Reviews a coffee
        coffeeReview.addScore(this, score);
    }

    public static void main(String[] args) {
        Person p1 = new Person("Person1");
        Person p2 = new Person("Person2");
        Person p3 = new Person("Person3");
        Person p4 = new Person("Person4");

        CoffeeReview coffeeReview = p1.addCoffee("Monday
            coffee");
        p2.reviewCoffee(coffeeReview, 6);
        p3.reviewCoffee(coffeeReview, 3);
        p4.reviewCoffee(coffeeReview, 5);
        System.out.println(p1.getAverageScore());
        /* Expected output: 5.5 */
    }
}
```

Listing D.5: Source code for study task 5

D.6 Task 6 - Comprehension

Related question: What is the output of this code?
Answer: 35.0

```
class A {
    public int b(double d1, double d2) {
        return (int)(d1 + d2);
    }
}

class B extends A {
    public double a(int i) {
        return i * 2;
    }

    @Override
    public int b(double d1, double d2) {
        return (int)(d1 / d2);
    }
}

class C extends B {
    @Override
    public int b(double d1, double d2) {
        return (int)(d1 * d2);
    }
    public List < Integer > d(int f) {
        return Arrays.asList(b(f, f), super.b(f + 0.5, f));
    }
}

class D extends C {
    private List < Integer > e;
    public D(int e) {
        this.e = d(e);
    }
    public double e() {
        e.stream().forEach(g -> a(g));
        return e.stream().reduce(0,
            Integer::sum).doubleValue();
    }
}

public class Task6 {
    public static void main(String[] args) {
        D d = new D(5);
        System.out.println(d.e());
    }
}
```

Listing D.6: Source code for study task 6

Appendix E

Grading scripts

E.1 Pretest grading script

```python
def grade_pretest(pretest):
    col_names = [(f'pretest.{i}-{1}',
        f'pretest.{i}-{1}_correct') for i in range(1,11)]
    pretest['no_correct'] = [None]*len(pretest)
    for col in col_names:
        pretest[f'{col[0]}_participant_correct'] =
            [None]*len(pretest)
    pretest['helpType'] = [None]*len(pretest)

    for i in range(len(pretest)):
        row = pretest.iloc[i]
        summ = 0
        username = row['username']
        for j, colnames in enumerate(col_names):
            helpType = usernames[usernames['username'] ==
                username]['helpType'].index
            pretest.loc[pretest['username'] == username,
                'helpType'] =
                usernames['helpType'][helpType].iloc[0]
            score = row[colnames[0]] == row[colnames[1]]
            pretest.loc[pretest['username'] ==
                row['username'],
                f'{colnames[0]}_participant_correct'] =
            int(score)
          summm += score
      pretest.loc[pretest['username'] == row['username'],
          'no_correct'] = float(summ)
   return pretest
```

Listing E.1: Script for automatically graing the pretest

E.2 Study grading script

```
def grade_study(df, col):
   inds = list(df.index)
   l = [0]*len(inds)

   random.shuffle(inds)

   for ind in inds:
      print("#"*40)
      print(f'# Now grading for {col.capitalize()}: \n')
      print('# Participant\'s answer:\n')
      print("#  " + str(df[col][ind]))
      print("#"*40)
      l[ind] = float(input('Grade: '))

   df[f'{col}_points'] = l
   return df
```

Listing E.2: Script for grading the study answers

Appendix F

Result and analysis supplement

F.1 Plots and p-values

F.1.1 Study scores

Table F.1 shows results from statistical tests done to find a difference in study scores between the conditions and between the expertise groups on each study task. The table presents χ^2 and p-values from Kruskal-Wallis tests. A Shapiro-Wilk test was performed to test for normality in these groups as well, which presented evidence of non-normality in study scores for both groups for every task.

Table F.1: P-values reported from Kruskal-Wallis tests for difference in study scores between the conditions and the expertise groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Task	**Conditions**		**Expertise**	
	χ^2	p	χ^2	p
1	0.125	.727	0.127	.727
2	0.140	.699	1.348	.246
3	0.440	.507	0.440	.507
4	0	1	0.517	.472
5	1.094	.296	1.094	.296
6	n/a	n/a	n/a	n/a

Table F.2 shows results from Kruskal-Wallis tests done to investigate a difference in study scores between conditions within each expertise group. Once again, the tables show χ^2 and p-values. In addition, Shapiro-Wilk tests for normality within each group presented evidence of non-normality for each group. Furthermore, it should be noted that the sample sizes were small, so the results should be taken with a grain of salt.

Table F.2: P-values reported from Kruskal-Wallis tests for difference in study scores between the conditions within expertise groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Task	**Low**		**High**	
	χ^2	p	χ^2	p
1	0.238	.626	0.004	.696
2	0.143	.706	0.152	.696
3	0.343	.558	2.268	.132
4	0.732	.392	0.850	.357
5	1.921	.166	0.004	.951
6	n/a	n/a	n/a	n/a

Table F.3 shows results from Kruskal-Wallis tests done to investigate a difference in study scores between conditions within each performance group. The tables show χ^2 and p-values. In addition, Shapiro-Wilk tests for normality within each group presented evidence of non-normality for each group. Once again, the sample sizes are small, and the results should therefore be taken with a grain of salt.

Table F.3: P-values reported from Kruskal-Wallis tests for difference in study scores between the conditions within performance groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Task	**Low**		**High**	
	χ^2	p	χ^2	p
1	0.250	.617	0.449	.503
2	n/a	n/a	0.049	.824
3	0.500	.480	0.007	.931
4	0.042	.838	0.260	.611
5	0.023	.880	1.347	.246
6	n/a	n/a	n/a	n/a

F.1.2 Number of fixations lines with bugs

Table F.4 shows results from statistical tests done to find a difference in number of fixations on lines with bugs between the conditions, expertise groups, and performance groups. The table presents χ^2 and p-values from Kruskal-Wallis tests.

Table F.4: P-values reported from Kruskal-Wallis tests for difference in number of fixations on bugs between the conditions, expertise groups and performance groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Bug	**Conditions**		**Expertise**		**Performance**	
	χ^2	p	χ^2	p	χ^2	p
Task1.Bug	0.329	.567	0.329	.567	1.665	.197
Task3.Bug1	0.026	.872	0.511	.475	0.447	.504
Task3.Bug2	1.848	.174	0.002	.963	0.005	.945
Task5.Bug1	0.002	.965	0.069	.792	0.031	.860
Task5.Bug2	4.906	.029	3.168	.075	2.399	.121

F.1.3 Time spent on lines with bugs

Table F.5 shows results from statistical tests done to find a difference in normalized fixation duration on lines with bugs between the conditions, expertise groups and performance groups. The table presents χ^2 and p-values from Kruskal-Wallis tests.

Table F.5: P-values reported from Kruskal-Wallis tests for difference in normalized fixation duration on bugs between the conditions, expertise groups and performance groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Bug	**Conditions**		**Expertise**		**Performance**	
	χ^2	p	χ^2	p	χ^2	p
Task1.Bug	0.025	.874	0.039	.843	1.930	.165
Task3.Bug1	0.103	.748	0.280	.597	0.080	.765
Task3.Bug2	1.724	.189	0.002	.963	0.009	.926
Task5.Bug1	0.840	.359	0.433	.511	1.508	.220
Task5.Bug2	0.756	.385	2.025	.155	.309	.578

F.1.4 Distance to lines with bugs

Table F.6 shows results from statistical tests done to find a difference percentile distance from lines with bugs on the tasks between the conditions, expertise groups and performance groups. The table presents χ^2 and p-values from Kruskal-Wallis tests.

Table F.6: P-values reported from Kruskal-Wallis tests for difference in distance from lines with bugs between the conditions, expertise groups and performance groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Percentile	**Conditions**		**Expertise**		**Performance**	
	χ^2	p	χ^2	p	χ^2	p
			Task 1			
0	0.003	.955	1.115	.291	2.578	.108
25	0.0128	.910	0.854	.355	1.161	.281
50	0.023	.880	0.259	.611	0.129	.720
75	0.461	.497	0.487	.485	0.461	.497
100	4.432	.035	1.210	.271	0.731	.393
			Task 3			
0	0.103	.749	0.036	.851	0.103	.748
25	0.751	.386	0.513	.474	0.437	.509
50	2.165	.141	0.889	.346	0.001	.970
75	0.888	.346	0.853	.356	0.104	.748
100	0.172	.679	0.142	.706	0.299	.584
			Task 5			
0	0.128	.720	1.114	.291	0.542	.461
25	0.103	.749	0.240	.624	0.173	.678
50	0.364	.547	1.365	.243	0.157	.692
75	0.157	.692	5.115	.024	0.388	.533
100	0.785	.376	2.162	.142	0.343	.559

Table F.7 below shows p-values and Pearson's r reported from Pearson correlation tests between total study scores and percent of first fixation after consulting help section that was on a line with a bug for all bugs.

Table F.7: P-values and Pearson's r reported from Person correlation between total study scores and percent of first fixation after consulting help section that was on a line with bug. Degrees of freedom were 14 for all bugs. All values are rounded to three decimal places.

Bug	r	p
Task1.Bug	-0.120	.660
Task3.Bug1	0.030	.911
Task3.Bug2	-0.294	.268
Task5.Bug1	-0.093	.732
Task5.Bug2	0.157	.561

F.1.5 Time spent per task

Table F.8 shows results from statistical tests done to find a difference in time per task between the conditions, expertise groups and performance groups. The table presents χ^2 and p-values from Kruskal-Wallis tests.

Table F.8: P-values reported from Kruskal-Wallis tests for difference in time spent per task between the conditions, expertise groups and performance groups. Degrees of freedom were 1 for all groups. All values are rounded to three decimal places.

Task	**Conditions**		**Expertise**		**Performance**	
	χ^2	p	χ^2	p	χ^2	p
1	1.740	.187	0.751	.386	1.078	.299
2	0.039	.843	0.352	.553	0.266	.606
3	0.460	.498	0.036	.851	0.927	.336
4	0.513	.474	0.070	.791	1.078	.299
5	2.273	.132	0.051	.821	6.497	.011
6	3.138	.077	0.036	.851	2.114	.146

F.2 Mean and standard deviation tables

This section contains a table presenting mean and standard deviation of number of fixations and duration of fixations on every AOI that was created for the conditions, expertise groups and performance groups. In addition, the table contains mean and standard deviation of the study scores and pretest scores for the different groups.

Table F.9: Mean and standard deviation for every AOI and variable in the study for the control and help group and high and low expertise and performance group. Variables followed by (d) is duration. Variables followed by (nf) is number of fixations.

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Study												
Task1 points	0.438	0.512	0.5	0.516	0.5	0.516	0.438	0.512	0.2	0.414	0.706	0.47
Task2 points	0.25	0.447	0.312	0.479	0.375	0.5	0.188	0.403	0.0	0.0	0.529	0.514
Task3 points	0.688	0.602	0.812	0.544	0.812	0.544	0.688	0.602	0.333	0.488	1.118	0.332
Task4 points	0.625	0.5	0.625	0.5	0.688	0.479	0.562	0.512	0.4	0.507	0.824	0.393
Task5 points	0.375	0.5	0.562	0.512	0.375	0.5	0.562	0.512	0.267	0.458	0.647	0.493
Task6 points	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Study total	2.375	1.455	2.812	1.642	2.75	1.807	2.438	1.263	1.2	0.775	3.824	0.809
Pretest												
Pretest1 points	0.125	0.342	0.188	0.403	0.125	0.342	0.188	0.403	0.2	0.414	0.118	0.332
Pretest2 points	0.5	0.516	0.5	0.516	0.375	0.5	0.625	0.5	0.6	0.507	0.412	0.507
Pretest3 points	0.188	0.403	0.375	0.5	0.188	0.403	0.375	0.5	0.267	0.458	0.294	0.47
Pretest4 points	0.438	0.512	0.312	0.479	0.25	0.447	0.5	0.516	0.267	0.458	0.471	0.514
Pretest5 points	0.125	0.342	0.312	0.479	0.125	0.342	0.312	0.479	0.133	0.352	0.294	0.47
Pretest6 points	0.188	0.403	0.438	0.512	0.062	0.25	0.562	0.512	0.4	0.507	0.235	0.437
Pretest7 points	0.625	0.5	0.75	0.447	0.562	0.512	0.812	0.403	0.6	0.507	0.765	0.437
Pretest8 points	0.312	0.479	0.375	0.5	0.125	0.342	0.562	0.512	0.4	0.507	0.294	0.47
Pretest9 points	0.812	0.403	0.75	0.447	0.625	0.5	0.938	0.25	0.733	0.458	0.824	0.393
Pretest10 points	0.375	0.5	0.5	0.516	0.188	0.403	0.688	0.479	0.467	0.516	0.412	0.507
Pretest total	3.688	1.662	4.5	1.862	2.625	0.719	5.562	1.209	4.067	1.668	4.118	1.933
Task1												
Task1.Bug (d)	0.052	0.052	0.052	0.057	0.045	0.035	0.059	0.068	0.04	0.051	0.062	0.055

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task1.Bug (nf)	70.188	54.699	97.333	91.375	89.625	68.532	76.6	82.684	68.357	78.626	95.647	71.323
Task1.Line1 (d)	0.002	0.003	0.001	0.001	0.002	0.002	0.001	0.001	0.002	0.002	0.001	0.001
Task1.Line1 (nf)	6.0	4.082	3.6	2.836	4.875	3.603	3.5	2.881	4.833	3.601	3.875	3.182
Task1.Line2 (d)	0.006	0.007	0.005	0.007	0.004	0.006	0.007	0.008	0.004	0.006	0.006	0.008
Task1.Line2 (nf)	9.727	11.01	9.733	11.43	9.0	9.936	10.727	12.807	7.545	8.383	11.333	12.67
Task1.Line3 (d)	0.009	0.011	0.012	0.012	0.01	0.01	0.011	0.012	0.01	0.012	0.011	0.011
Task1.Line3 (nf)	16.929	21.833	22.0	18.872	21.643	20.805	16.923	20.139	19.833	24.616	19.0	16.848
Task1.Line4 (d)	0.007	0.007	0.007	0.009	0.007	0.007	0.008	0.01	0.007	0.008	0.008	0.009
Task1.Line4 (nf)	12.615	13.118	15.067	15.36	15.571	14.383	12.286	14.264	15.0	16.717	13.125	12.414
Task1.Line8 (d)	0.013	0.015	0.007	0.006	0.009	0.011	0.011	0.014	0.009	0.013	0.011	0.011
Task1.Line8 (nf)	17.357	16.439	16.231	21.576	18.812	23.224	13.909	9.279	15.0	21.552	18.5	16.294
Task1.Line9 (d)	0.012	0.012	0.005	0.003	0.009	0.011	0.008	0.008	0.006	0.007	0.01	0.011
Task1.Line9 (nf)	15.846	17.597	11.4	15.138	18.167	19.697	9.273	10.808	11.5	15.05	15.769	17.674
Task1.Line10 (d)	0.004	0.002	0.001	0.001	0.002	0.002	0.002	0.003	0.001	0.002	0.002	0.003
Task1.Line10 (nf)	8.0	7.842	3.0	4.435	5.429	6.655	4.6	6.504	3.667	4.719	6.5	7.765
Task1.Line11 (d)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Task1.Line11 (nf)	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
Task1.Line12 (d)	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.001	nan	0.0	0.0
Task1.Line12 (nf)	1.0	0.0	1.5	0.707	1.5	0.707	1.0	0.0	2.0	nan	1.0	0.0
Task1.Line14 (d)	0.006	0.01	0.004	0.004	0.004	0.003	0.006	0.01	0.003	0.004	0.007	0.01
Task1.Line14 (nf)	11.667	14.291	7.929	5.89	9.0	7.246	10.562	13.574	7.5	7.24	12.067	13.572
Task1.Line15 (d)	0.005	0.007	0.005	0.004	0.004	0.003	0.005	0.007	0.004	0.003	0.005	0.007
Task1.Line15 (nf)	9.267	10.8	12.462	10.121	11.769	11.211	9.867	9.999	9.769	7.407	11.6	12.682
Task1.Line16 (d)	0.004	0.004	0.006	0.004	0.005	0.004	0.005	0.004	0.005	0.005	0.005	0.003
Task1.Line16 (nf)	9.077	9.087	12.462	9.854	14.333	12.003	7.714	5.283	11.385	12.868	10.154	4.413

Continued on next page

Table F.9 – continued from previous page

Variable	Condition				Expertise				Performance			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task1.Line17 (d)	0.004	0.003	0.005	0.003	0.005	0.004	0.004	0.003	0.004	0.003	0.004	0.003
Task1.Line17 (nf)	6.923	6.664	12.583	8.959	12.5	9.539	7.0	5.958	10.333	9.838	9.0	6.708
Task1.Line18 (d)	0.005	0.003	0.005	0.005	0.005	0.003	0.005	0.005	0.004	0.004	0.005	0.004
Task1.Line18 (nf)	9.4	7.462	9.643	7.571	10.214	6.941	8.867	7.954	8.538	8.006	10.312	6.993
Task1.Line19 (d)	0.001	0.001	0.0	0.0	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Task1.Line19 (nf)	1.6	1.342	2.0	1.414	2.2	1.643	1.4	0.894	1.5	1.225	2.25	1.5
Task1.Line20 (d)	0.007	0.006	0.003	0.002	0.006	0.005	0.005	0.005	0.004	0.004	0.006	0.006
Task1.Line20 (nf)	11.062	8.637	5.923	4.051	10.643	7.541	7.0	6.918	6.071	4.141	11.267	8.811
Task1.Line21 (d)	0.003	nan	n/a	n/a	n/a	n/a	0.003	nan	n/a	n/a	0.003	nan
Task1.Line21 (nf)	4.0	nan	nan	nan	nan	nan	4.0	nan	nan	nan	4.0	nan
Task1.Line22 (d)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Task1.Line22 (nf)	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
Task1.Code (d)	0.113	0.083	0.072	0.067	0.08	0.063	0.105	0.089	0.07	0.068	0.113	0.081
Task1.Code (nf)	178.812	132.179	152.688	145.458	158.312	110.026	173.188	163.614	120.267	75.317	205.882	167.159
Task1.Help (d)	0.001	0.0	0.113	0.058	0.083	0.068	0.115	0.064	0.105	0.052	0.097	0.077
Task1.Help (nf)	2.0	1.414	162.667	86.597	133.5	97.796	152.889	101.244	164.571	96.067	129.2	100.01
Task1.HelpButton (d)	0.003	0.002	0.006	0.007	0.003	0.002	0.006	0.007	0.004	0.002	0.005	0.007
Task1.HelpButton (nf)	3.462	1.561	10.333	5.08	6.0	5.463	8.286	4.762	7.308	4.939	7.0	5.516
Task1.TaskType (d)	0.004	0.003	0.002	0.002	0.003	0.003	0.003	0.002	0.003	0.002	0.003	0.003
Task1.TaskType (nf)	4.833	3.07	3.714	1.978	4.462	3.072	4.0	2.0	4.417	1.782	4.071	3.125
Task1.Question (d)	0.022	0.02	0.012	0.011	0.018	0.021	0.017	0.012	0.02	0.019	0.014	0.014
Task1.Question (nf)	24.25	16.139	20.867	17.167	21.067	14.602	24.062	18.379	22.533	15.179	22.688	18.066
Task2												
Task2.Line1 (d)	0.001	0.001	0.002	0.002	0.001	0.001	0.002	0.002	0.002	0.001	0.001	0.002

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task2.Line1 (nf)	4.0	3.317	2.143	1.676	3.667	3.077	2.167	1.835	5.25	2.872	1.75	1.389
Task2.Line2 (d)	0.004	0.002	0.007	0.006	0.004	0.004	0.008	0.005	0.004	0.003	0.006	0.006
Task2.Line2 (nf)	6.6	4.142	9.0	7.969	6.333	5.193	10.143	7.403	8.857	5.61	7.083	6.653
Task2.Line3 (d)	0.008	0.012	0.014	0.019	0.006	0.005	0.015	0.021	0.007	0.007	0.013	0.019
Task2.Line3 (nf)	11.786	13.791	15.0	15.875	13.077	13.883	13.333	15.79	13.7	15.777	12.867	14.172
Task2.Line4 (d)	0.01	0.011	0.011	0.014	0.009	0.01	0.012	0.015	0.009	0.009	0.012	0.014
Task2.Line4 (nf)	17.0	22.372	15.462	15.447	17.857	21.271	14.333	16.3	19.8	23.328	14.0	15.875
Task2.Line5 (d)	0.02	0.022	0.02	0.02	0.02	0.021	0.02	0.022	0.022	0.018	0.019	0.022
Task2.Line5 (nf)	32.769	38.14	28.846	24.711	34.643	36.58	26.333	25.303	40.8	40.947	24.562	23.341
Task2.Line6 (d)	0.013	0.015	0.013	0.011	0.013	0.014	0.012	0.012	0.015	0.013	0.012	0.013
Task2.Line6 (nf)	22.692	26.515	19.462	16.656	24.214	26.6	17.417	14.576	28.6	28.613	16.375	15.375
Task2.Line7 (d)	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.003	0.002	0.002
Task2.Line7 (nf)	3.75	1.669	4.143	5.273	4.778	4.265	2.667	2.251	4.375	4.809	3.429	1.902
Task2.Line8 (d)	0.017	0.017	0.026	0.017	0.022	0.018	0.021	0.018	0.019	0.018	0.023	0.018
Task2.Line8 (nf)	27.067	27.249	35.846	18.645	34.429	26.555	27.857	20.817	34.154	28.847	28.533	18.704
Task2.Line9 (d)	0.022	0.022	0.023	0.022	0.023	0.02	0.022	0.024	0.02	0.02	0.025	0.023
Task2.Line9 (nf)	30.312	29.35	34.462	23.599	34.857	27.768	29.667	26.062	34.538	30.344	30.25	23.87
Task2.Line10 (d)	0.034	0.027	0.021	0.021	0.028	0.022	0.027	0.027	0.026	0.024	0.029	0.026
Task2.Line10 (nf)	47.867	40.055	37.857	30.168	43.214	31.271	42.867	39.916	46.769	39.218	40.0	32.898
Task2.Line11 (d)	0.009	0.006	0.002	0.002	0.006	0.005	0.003	0.005	0.003	0.003	0.005	0.006
Task2.Line11 (nf)	13.8	11.432	4.0	4.848	8.333	5.046	6.875	11.243	7.0	5.831	7.875	10.999
Task2.Line12 (d)	0.002	0.001	0.001	0.0	0.002	0.002	0.001	0.001	0.001	0.001	0.002	0.001
Task2.Line12 (nf)	3.143	1.952	1.5	0.577	3.167	2.137	1.8	0.837	2.2	2.168	2.833	1.472
Task2.Line13 (d)	0.01	0.016	0.007	0.007	0.006	0.008	0.01	0.015	0.005	0.007	0.011	0.015
Task2.Line13 (nf)	16.067	28.369	17.786	32.957	9.714	9.417	23.6	40.408	17.0	34.658	16.812	27.073

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task2.Line14 (d)	n/a	n/a	0.0	nan	n/a	n/a	0.0	nan	n/a	n/a	0.0	nan
Task2.Line14 (nf)	nan	nan	1.0	nan	nan	nan	1.0	nan	nan	nan	1.0	nan
Task2.Line15 (d)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Task2.Line15 (nf)	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
Task2.Code (d)	0.117	0.104	0.11	0.071	0.112	0.088	0.115	0.092	0.095	0.081	0.13	0.094
Task2.Code (nf)	187.375	182.393	198.286	200.345	178.643	130.901	204.562	230.061	209.643	230.95	177.438	146.186
Task2.Help (d)	n/a	n/a	0.084	0.069	0.022	0.02	0.101	0.068	0.075	0.048	0.094	0.097
Task2.Help (nf)	nan	nan	114.556	84.513	60.0	38.184	130.143	89.47	127.8	92.556	98.0	83.467
Task2.HelpButton (d)	0.003	0.003	0.004	0.003	0.003	0.003	0.004	0.004	0.005	0.004	0.003	0.002
Task2.HelpButton (nf)	3.385	2.434	6.583	4.209	4.692	3.119	5.167	4.387	6.0	4.099	4.071	3.269
Task2.TaskType (d)	0.007	0.007	0.005	0.008	0.007	0.008	0.005	0.008	0.006	0.009	0.006	0.007
Task2.TaskType (nf)	8.333	6.683	11.692	25.49	9.154	6.817	10.533	23.799	12.429	24.513	7.357	6.259
Task2.Question (d)	0.013	0.02	0.005	0.004	0.006	0.006	0.012	0.02	0.012	0.016	0.007	0.014
Task2.Question (nf)	11.667	11.056	8.667	8.191	10.533	10.169	9.8	9.511	15.0	12.616	6.471	4.125
Task3												
Task3.Bug1 (d)	0.027	0.068	0.008	0.009	0.028	0.07	0.008	0.009	0.027	0.07	0.008	0.011
Task3.Bug1 (nf)	51.143	95.08	31.786	31.747	56.769	97.558	28.2	30.883	55.923	100.728	28.933	21.416
Task3.Bug2 (d)	0.009	0.008	0.014	0.011	0.011	0.009	0.012	0.011	0.012	0.012	0.01	0.008
Task3.Bug2 (nf)	33.467	38.255	48.308	43.043	42.643	45.864	38.071	35.909	41.833	43.401	39.25	39.572
Task3.Line1 (d)	0.001	0.001	0.002	0.002	0.001	0.001	0.002	0.002	0.003	0.002	0.001	0.001
Task3.Line1 (nf)	5.667	5.788	7.25	6.519	5.455	5.592	8.167	6.853	11.5	9.256	4.846	3.913
Task3.Line2 (d)	0.0	0.0	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Task3.Line2 (nf)	3.0	2.608	4.143	3.288	3.857	2.968	3.333	3.141	5.25	3.862	2.889	2.315
Task3.Line3 (d)	0.003	0.003	0.004	0.005	0.004	0.004	0.003	0.005	0.006	0.006	0.002	0.002

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task3.Line3 (nf)	9.636	11.03	14.364	17.506	12.364	10.529	11.636	18.145	16.875	20.343	9.214	9.641
Task3.Line4 (d)	0.003	0.003	0.004	0.005	0.003	0.002	0.004	0.005	0.004	0.003	0.003	0.004
Task3.Line4 (nf)	11.5	10.395	13.091	15.675	10.083	9.605	15.333	16.904	13.375	12.42	11.692	13.99
Task3.Line5 (d)	0.004	0.004	0.004	0.004	0.004	0.003	0.005	0.004	0.005	0.003	0.004	0.004
Task3.Line5 (nf)	14.1	13.051	16.455	15.384	15.182	11.957	15.5	16.662	16.625	13.989	14.538	14.541
Task3.Line6 (d)	0.001	0.0	0.0	0.0	0.001	0.0	0.0	0.0	0.001	0.0	0.0	0.0
Task3.Line6 (nf)	1.6	0.894	2.0	1.0	2.0	0.894	1.0	0.0	1.667	0.577	1.8	1.095
Task3.Line7 (d)	0.007	0.005	0.009	0.008	0.007	0.004	0.01	0.009	0.007	0.005	0.008	0.008
Task3.Line7 (nf)	26.917	20.188	30.846	21.424	27.538	19.432	30.5	22.363	27.9	20.475	29.667	21.205
Task3.Line8 (d)	0.007	0.005	0.013	0.011	0.01	0.009	0.01	0.01	0.011	0.01	0.01	0.009
Task3.Line8 (nf)	28.5	25.732	43.769	32.484	37.077	30.554	34.714	29.873	35.636	30.897	36.0	29.77
Task3.Line10 (d)	0.008	0.006	0.008	0.006	0.008	0.006	0.008	0.007	0.009	0.007	0.007	0.006
Task3.Line10 (nf)	31.455	31.297	26.846	21.404	31.333	29.478	26.583	22.813	34.889	31.227	25.4	22.538
Task3.Line11 (d)	0.0	0.0	0.001	0.001	0.001	0.001	0.001	0.0	0.001	0.0	0.001	0.001
Task3.Line11 (nf)	2.571	3.309	3.143	2.41	2.667	2.291	3.2	3.834	2.167	1.941	3.375	3.335
Task3.Line12 (d)	0.011	0.009	0.008	0.009	0.009	0.007	0.01	0.01	0.009	0.006	0.01	0.01
Task3.Line12 (nf)	36.154	31.469	29.0	23.402	37.231	34.332	27.923	18.441	35.2	32.241	30.938	24.928
Task3.Line13 (d)	0.012	0.014	0.009	0.007	0.01	0.008	0.01	0.013	0.009	0.008	0.011	0.012
Task3.Line13 (nf)	36.286	30.706	33.786	23.577	38.077	27.011	32.4	27.448	32.417	26.349	37.0	27.983
Task3.Line14 (d)	0.009	0.008	0.009	0.006	0.008	0.006	0.01	0.007	0.008	0.005	0.01	0.008
Task3.Line14 (nf)	29.0	25.109	32.692	19.855	29.231	23.555	32.214	22.015	26.909	20.873	33.438	23.642
Task3.Line15 (d)	0.006	0.005	0.007	0.006	0.007	0.006	0.007	0.006	0.006	0.005	0.008	0.006
Task3.Line15 (nf)	25.071	27.43	25.5	18.003	27.0	27.209	23.8	18.974	23.25	26.626	26.812	20.183
Task3.Line16 (d)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task3.Line16 (nf)	1.6	0.894	1.6	0.894	1.714	0.951	1.333	0.577	1.4	0.894	1.8	0.837

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task3.Line17 (d)	0.006	0.005	0.008	0.007	0.005	0.005	0.008	0.007	0.007	0.005	0.006	0.007
Task3.Line17 (nf)	19.786	14.327	26.5	19.394	22.077	16.621	24.067	17.99	26.0	21.388	21.0	13.342
Task3.Line18 (d)	0.007	0.004	0.012	0.009	0.009	0.008	0.009	0.007	0.008	0.006	0.01	0.009
Task3.Line18 (nf)	25.643	17.403	46.143	39.559	40.25	38.714	32.625	26.252	32.077	27.936	39.2	35.36
Task3.Line19 (d)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Task3.Line19 (nf)	1.5	0.707	1.5	0.707	1.5	0.707	1.5	0.707	1.5	0.707	1.5	0.707
Task3.Line20 (d)	0.005	0.003	0.006	0.004	0.005	0.003	0.005	0.004	0.007	0.004	0.004	0.002
Task3.Line20 (nf)	20.75	16.069	24.846	19.373	23.091	13.816	22.714	20.641	30.545	23.032	16.857	8.725
Task3.Line21 (d)	0.006	0.007	0.01	0.009	0.007	0.006	0.009	0.009	0.01	0.008	0.007	0.008
Task3.Line21 (nf)	29.714	41.292	38.929	33.047	30.769	38.53	37.4	36.68	47.333	47.418	24.562	24.039
Task3.Line22 (d)	0.004	0.005	0.005	0.005	0.005	0.005	0.005	0.005	0.006	0.006	0.004	0.004
Task3.Line22 (nf)	15.583	18.178	19.923	14.39	16.0	14.471	19.286	17.709	23.0	18.788	13.786	12.963
Task3.Line23 (d)	0.0	0.0	0.0	0.0	0.0	nan	0.0	0.0	0.0	nan	0.0	0.0
Task3.Line23 (nf)	1.5	0.707	2.5	0.707	3.0	nan	1.667	0.577	1.0	nan	2.333	0.577
Task3.Line24 (d)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	nan	0.0	nan	0.0	0.0
Task3.Line24 (nf)	1.0	0.0	1.0	0.0	1.0	0.0	1.0	nan	1.0	nan	1.0	0.0
Task3.Line25 (d)	0.002	0.001	0.003	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
Task3.Line25 (nf)	5.846	3.579	10.538	8.922	6.75	5.895	9.429	7.959	7.273	5.587	8.867	8.123
Task3.Line26 (d)	0.002	0.002	0.005	0.004	0.004	0.003	0.004	0.004	0.004	0.003	0.003	0.004
Task3.Line26 (nf)	7.769	5.341	16.538	13.37	11.25	7.641	12.929	13.379	12.455	6.455	11.933	13.546
Task3.Line27 (d)	0.002	0.003	0.005	0.004	0.003	0.003	0.004	0.004	0.004	0.004	0.003	0.003
Task3.Line27 (nf)	7.0	7.539	15.167	9.36	9.5	6.789	12.231	11.211	11.3	7.04	10.667	10.735
Task3.Line28 (d)	0.003	0.004	0.004	0.003	0.003	0.004	0.004	0.003	0.005	0.004	0.002	0.002
Task3.Line28 (nf)	8.857	7.513	14.0	6.916	10.462	6.96	12.143	8.254	13.818	6.838	9.625	7.762
Task3.Line29 (d)	0.002	0.003	0.003	0.003	0.003	0.003	0.003	0.002	0.003	0.003	0.002	0.002

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task3.Line29 (nf)	7.455	5.447	9.769	5.747	8.4	4.695	8.929	6.354	9.091	5.009	8.385	6.265
Task3.Line30 (d)	0.002	0.002	0.002	0.001	0.002	0.001	0.002	0.002	0.002	0.002	0.001	0.001
Task3.Line30 (nf)	6.083	4.907	7.417	4.926	6.545	4.803	6.923	5.09	7.0	5.31	6.538	4.648
Task3.Line31 (d)	0.001	0.001	0.002	0.002	0.002	0.002	0.002	0.001	0.002	0.002	0.001	0.001
Task3.Line31 (nf)	5.083	4.814	8.333	5.105	6.0	4.382	7.308	5.793	7.455	5.628	6.077	4.804
Task3.Line32 (d)	n/a	n/a	0.0	0.0	0.001	0.001	0.0	0.0	0.0	nan	0.0	0.0
Task3.Line32 (nf)	nan	nan	1.8	1.304	3.0	1.414	1.0	0.0	1.0	nan	2.0	1.414
Task3.Line33 (d)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Task3.Line33 (nf)	2.917	2.275	6.091	3.562	3.636	3.776	5.167	2.791	4.111	2.369	4.643	3.875
Task3.Line34 (d)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.001	0.001	0.001	0.002
Task3.Line34 (nf)	4.7	2.983	5.692	5.313	5.273	5.274	5.25	3.646	4.583	2.712	6.0	5.762
Task3.Line35 (d)	0.0	nan	0.001	0.001	0.001	0.001	n/a	n/a	n/a	n/a	0.001	0.001
Task3.Line35 (nf)	1.0	nan	2.0	1.414	1.667	1.155	nan	nan	nan	nan	1.667	1.155
Task3.Line36 (d)	0.001	0.001	0.001	0.001	0.001	0.002	0.001	0.001	0.001	0.002	0.001	0.001
Task3.Line36 (nf)	3.333	2.015	4.833	3.186	4.0	3.59	4.143	2.033	3.636	2.203	4.462	3.126
Task3.Line37 (d)	0.001	0.001	0.001	0.0	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.0
Task3.Line37 (nf)	2.182	1.168	2.9	1.729	2.667	1.5	2.417	1.505	2.333	1.5	2.667	1.497
Task3.Line38 (d)	0.0	nan	0.0	0.0	0.0	0.0	0.0	0.0	0.001	0.0	0.0	0.0
Task3.Line38 (nf)	1.0	nan	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0	1.0	0.0
Task3.Line39 (d)	0.001	0.002	0.002	0.002	0.002	0.003	0.001	0.002	0.002	0.003	0.001	0.002
Task3.Line39 (nf)	4.25	3.494	6.545	3.56	6.182	4.4	4.583	2.746	5.0	3.435	5.667	3.939
Task3.Line40 (d)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Task3.Line40 (nf)	3.692	2.25	5.154	3.625	4.083	2.429	4.714	3.561	3.091	2.119	5.4	3.312
Task3.Line41 (d)	0.001	0.001	0.0	0.0	0.0	0.0	0.001	0.001	0.0	0.0	0.0	0.001
Task3.Line41 (nf)	2.0	2.236	1.4	0.548	1.0	0.0	2.0	1.826	1.5	0.707	1.75	1.753

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task3.Line42 (d)	0.0	0.0	0.001	0.0	0.001	0.0	0.0	0.0	0.001	0.0	0.0	0.0
Task3.Line42 (nf)	1.0	0.0	2.0	1.0	1.5	0.837	1.5	1.0	1.5	1.0	1.5	0.837
Task3.Line43 (d)	0.011	0.012	0.012	0.008	0.008	0.005	0.014	0.013	0.011	0.01	0.012	0.011
Task3.Line43 (nf)	34.923	28.031	48.857	34.561	37.833	28.97	45.6	34.467	42.167	38.544	42.133	26.624
Task3.Line44 (d)	0.012	0.01	0.015	0.009	0.014	0.009	0.013	0.01	0.01	0.009	0.015	0.009
Task3.Line44 (nf)	39.5	37.067	58.857	43.903	56.25	45.349	43.875	38.184	44.692	43.231	53.067	40.202
Task3.Line45 (d)	0.01	0.014	0.012	0.007	0.014	0.014	0.009	0.008	0.009	0.008	0.013	0.013
Task3.Line45 (nf)	30.867	36.144	50.154	37.932	48.615	43.735	32.2	30.841	36.615	39.517	42.6	36.992
Task3.Line47 (d)	0.001	0.0	0.002	0.002	0.001	0.002	0.001	0.001	0.002	0.002	0.001	0.0
Task3.Line47 (nf)	2.375	1.847	6.429	6.997	3.875	5.768	4.714	4.889	7.6	8.234	2.6	1.647
Task3.Line48 (d)	0.006	0.006	0.005	0.005	0.003	0.003	0.007	0.007	0.005	0.007	0.005	0.005
Task3.Line48 (nf)	17.533	16.186	22.4	22.398	13.214	16.428	25.875	20.271	18.0	21.075	21.688	18.245
Task3.Line49 (d)	0.005	0.005	0.006	0.007	0.004	0.005	0.007	0.007	0.006	0.008	0.005	0.004
Task3.Line49 (nf)	16.867	15.113	22.0	24.372	15.4	19.892	23.375	20.461	18.429	21.897	20.412	19.442
Task3.Line50 (d)	0.007	0.008	0.007	0.006	0.005	0.005	0.008	0.008	0.007	0.008	0.006	0.005
Task3.Line50 (nf)	22.133	21.384	25.467	25.746	22.429	23.78	25.0	23.614	23.143	25.678	24.375	21.881
Task3.Line51 (d)	0.006	0.003	0.004	0.004	0.004	0.002	0.006	0.004	0.004	0.003	0.005	0.003
Task3.Line51 (nf)	21.455	17.294	18.438	19.012	17.462	16.761	21.714	19.57	15.667	18.652	22.867	17.525
Task3.Line52 (d)	0.003	0.004	0.007	0.009	0.005	0.009	0.005	0.005	0.006	0.01	0.005	0.004
Task3.Line52 (nf)	15.091	17.621	19.333	15.375	16.0	15.725	18.857	16.997	14.75	15.702	19.929	16.74
Task3.Line53 (d)	0.003	0.004	0.004	0.003	0.003	0.003	0.006	0.004	0.005	0.005	0.003	0.003
Task3.Line53 (nf)	14.111	15.087	14.5	13.99	8.182	8.146	21.1	16.455	14.875	13.664	14.0	14.9
Task3.Line54 (d)	0.002	0.002	0.002	0.002	0.001	0.002	0.003	0.002	0.001	0.002	0.002	0.002
Task3.Line54 (nf)	10.0	12.987	8.333	10.084	7.0	9.826	11.6	12.799	8.0	10.235	9.846	12.233
Task3.Line55 (d)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task3.Line55 (nf)	1.5	0.707	1.25	0.5	1.0	0.0	1.667	0.577	1.0	0.0	1.5	0.577
Task3.Line56 (d)	0.0	0.0	0.0	nan	0.0	nan	0.0	0.0	0.0	0.0	0.0	nan
Task3.Line56 (nf)	1.0	0.0	1.0	nan	1.0	nan	1.0	0.0	1.0	0.0	1.0	nan
Task3.Code (d)	0.119	0.064	0.109	0.065	0.1	0.061	0.126	0.066	0.11	0.073	0.117	0.057
Task3.Code (nf)	422.467	309.737	437.312	288.91	417.467	341.57	442.0	252.827	428.429	364.148	431.529	233.577
Task3.Help (d)	0.0	0.0	0.135	0.123	0.143	0.165	0.089	0.093	0.143	0.156	0.084	0.09
Task3.Help (nf)	1.0	0.0	431.308	272.55	369.833	299.468	339.1	314.606	374.143	355.81	332.333	268.386
Task3.HelpButton (d)	0.001	0.001	0.002	0.002	0.002	0.002	0.001	0.001	0.002	0.002	0.001	0.001
Task3.HelpButton (nf)	4.833	4.174	4.846	3.078	6.583	3.988	3.231	2.242	4.4	2.914	5.133	4.015
Task3.TaskType (d)	0.004	0.005	0.001	0.001	0.002	0.002	0.003	0.005	0.002	0.004	0.003	0.004
Task3.TaskType (nf)	5.667	5.867	3.7	2.791	5.2	3.645	4.417	5.616	4.2	4.78	5.25	4.845
Task3.Question (d)	0.011	0.012	0.007	0.006	0.011	0.012	0.007	0.007	0.009	0.008	0.009	0.012
Task3.Question (nf)	24.143	23.452	23.357	18.866	26.538	22.762	21.333	19.595	19.308	15.734	27.6	24.395
						Task4						
Task4.Line1 (d)	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.002	0.002	0.001	0.001
Task4.Line1 (nf)	1.333	0.577	1.667	1.118	1.5	0.837	1.667	1.211	1.75	1.5	1.5	0.756
Task4.Line2 (d)	0.002	0.002	0.004	0.003	0.004	0.003	0.002	0.002	0.004	0.003	0.003	0.003
Task4.Line2 (nf)	2.4	1.517	3.625	2.825	3.75	2.712	2.2	1.643	3.5	2.38	3.0	2.55
Task4.Line3 (d)	0.002	0.002	0.005	0.006	0.004	0.005	0.003	0.004	0.004	0.002	0.004	0.005
Task4.Line3 (nf)	2.889	1.453	4.5	4.275	4.333	3.937	3.2	2.658	3.429	1.718	3.917	3.988
Task4.Line5 (d)	0.008	0.01	0.005	0.008	0.007	0.01	0.006	0.008	0.009	0.012	0.005	0.007
Task4.Line5 (nf)	5.818	5.964	5.25	6.032	5.929	7.0	4.889	3.79	7.375	7.308	4.533	4.941
Task4.Line6 (d)	0.009	0.009	0.005	0.005	0.007	0.005	0.007	0.01	0.011	0.01	0.004	0.003
Task4.Line6 (nf)	7.818	6.385	4.727	3.438	6.385	5.331	6.111	5.442	8.333	6.764	4.846	3.508

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task4.Line7 (d)	0.003	0.003	0.002	0.001	0.003	0.002	0.002	0.002	0.003	nan	0.003	0.002
Task4.Line7 (nf)	2.8	3.033	1.667	0.577	1.75	0.957	3.0	3.367	2.0	nan	2.429	2.573
Task4.Line9 (d)	0.015	0.011	0.004	0.004	0.01	0.012	0.008	0.008	0.01	0.012	0.008	0.008
Task4.Line9 (nf)	12.7	9.499	4.769	3.723	8.545	7.607	7.917	8.273	8.222	6.996	8.214	8.514
Task4.Line10 (d)	0.015	0.01	0.004	0.003	0.013	0.01	0.005	0.006	0.009	0.011	0.009	0.007
Task4.Line10 (nf)	10.889	5.011	4.667	2.674	9.111	3.756	6.0	5.36	6.25	5.203	8.0	4.778
Task4.Line11 (d)	0.004	0.001	0.002	0.001	0.002	0.001	0.003	0.002	0.001	0.001	0.003	0.001
Task4.Line11 (nf)	3.667	3.055	1.833	0.753	1.25	0.5	3.4	2.074	1.75	0.957	3.0	2.345
Task4.Line12 (d)	0.001	nan	n/a	n/a	n/a	n/a	0.001	nan	n/a	n/a	0.001	nan
Task4.Line12 (nf)	3.0	nan	nan	nan	nan	nan	3.0	nan	nan	nan	3.0	nan
Task4.Line14 (d)	0.006	0.004	0.004	0.003	0.006	0.005	0.004	0.003	0.005	0.005	0.005	0.003
Task4.Line14 (nf)	6.083	4.833	6.143	4.769	5.769	4.323	6.462	5.206	6.5	5.523	5.875	4.288
Task4.Line15 (d)	0.006	0.004	0.008	0.005	0.007	0.004	0.007	0.005	0.008	0.005	0.007	0.005
Task4.Line15 (nf)	5.643	4.088	8.583	5.071	6.167	3.738	7.714	5.455	7.0	4.05	7.0	5.292
Task4.Line16 (d)	0.008	0.006	0.008	0.006	0.007	0.004	0.009	0.007	0.01	0.007	0.006	0.005
Task4.Line16 (nf)	8.077	6.5	7.769	5.434	6.357	4.396	9.75	6.982	10.1	6.54	6.562	5.164
Task4.Line18 (d)	0.011	0.007	0.008	0.006	0.01	0.007	0.009	0.006	0.01	0.007	0.009	0.006
Task4.Line18 (nf)	9.643	6.134	9.2	6.259	8.643	4.717	10.133	7.239	8.75	5.723	9.882	6.47
Task4.Line19 (d)	0.011	0.007	0.01	0.007	0.01	0.007	0.01	0.007	0.012	0.007	0.009	0.008
Task4.Line19 (nf)	10.467	7.954	9.167	5.606	9.214	6.363	10.615	7.654	9.833	5.391	9.933	8.119
Task4.Line20 (d)	0.012	0.009	0.009	0.012	0.011	0.01	0.011	0.012	0.011	0.014	0.01	0.008
Task4.Line20 (nf)	11.333	9.123	10.286	10.542	8.615	5.881	12.625	11.798	9.538	10.236	11.875	9.387
Task4.Line21 (d)	0.007	0.007	0.002	0.001	0.003	0.003	0.004	0.005	0.001	0.001	0.004	0.005
Task4.Line21 (nf)	7.5	9.713	2.25	1.035	2.25	1.258	4.875	6.978	1.667	1.155	4.778	6.515
Task4.Line23 (d)	0.007	0.01	0.006	0.006	0.005	0.007	0.008	0.009	0.003	0.002	0.01	0.01

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task4.Line23 (nf)	7.143	8.865	7.077	7.421	5.417	5.791	8.467	9.456	3.5	2.355	10.0	9.783
Task4.Line24 (d)	0.008	0.007	0.009	0.009	0.009	0.008	0.009	0.009	0.008	0.008	0.01	0.008
Task4.Line24 (nf)	8.533	7.18	10.571	9.621	7.769	7.224	10.938	9.154	8.154	7.787	10.625	8.884
Task4.Line25 (d)	0.01	0.009	0.011	0.012	0.011	0.008	0.011	0.012	0.011	0.012	0.01	0.008
Task4.Line25 (nf)	10.067	11.126	11.417	10.211	8.667	5.105	12.267	13.414	9.083	10.157	11.933	11.029
Task4.Line26 (d)	0.005	0.005	0.004	0.005	0.003	0.004	0.005	0.005	0.003	0.004	0.005	0.005
Task4.Line26 (nf)	8.5	11.705	2.333	2.066	2.333	1.751	8.5	11.79	2.333	2.309	5.857	9.008
Task4.Line28 (d)	0.015	0.009	0.016	0.01	0.015	0.009	0.016	0.01	0.013	0.008	0.017	0.01
Task4.Line28 (nf)	15.5	9.818	19.267	13.766	14.0	6.047	20.438	15.002	14.214	11.735	19.882	11.651
Task4.Line29 (d)	0.02	0.014	0.02	0.017	0.02	0.016	0.02	0.015	0.019	0.013	0.021	0.017
Task4.Line29 (nf)	16.312	10.339	21.429	15.912	16.571	10.501	20.562	15.358	17.571	13.478	19.688	13.405
Task4.Line30 (d)	0.025	0.019	0.023	0.016	0.025	0.02	0.023	0.016	0.029	0.017	0.02	0.017
Task4.Line30 (nf)	18.625	12.317	24.857	20.869	18.533	13.087	24.533	19.928	22.429	20.44	20.75	13.601
Task4.Line31 (d)	0.017	0.013	0.008	0.006	0.017	0.014	0.01	0.006	0.015	0.011	0.011	0.011
Task4.Line31 (nf)	14.0	9.409	9.333	6.894	13.067	10.327	10.5	6.419	11.071	7.184	12.294	9.616
Task4.Line32 (d)	0.002	0.0	0.001	nan	0.002	nan	0.001	0.001	0.001	0.001	0.002	0.0
Task4.Line32 (nf)	1.333	0.577	1.0	nan	1.0	nan	1.333	0.577	1.0	0.0	1.5	0.707
Task4.Line33 (d)	0.0	nan	n/a	n/a	n/a	n/a	0.0	nan	n/a	n/a	0.0	nan
Task4.Line33 (nf)	1.0	nan	nan	nan	nan	nan	1.0	nan	nan	nan	1.0	nan
Task4.Code (d)	0.146	0.103	0.096	0.057	0.113	0.085	0.128	0.089	0.111	0.088	0.129	0.086
Task4.Code (nf)	133.812	102.449	137.625	151.695	104.25	68.724	167.188	163.338	114.467	115.589	154.471	137.595
Task4.Help (d)	n/a	n/a	0.11	0.101	0.107	0.13	0.112	0.08	0.073	0.07	0.133	0.115
Task4.Help (nf)	nan	nan	86.846	64.648	69.167	68.93	102.0	61.795	55.2	35.759	106.625	72.604
Task4.HelpButton (d)	0.004	0.003	0.004	0.003	0.004	0.002	0.004	0.004	0.005	0.004	0.003	0.002
Task4.HelpButton (nf)	3.167	1.642	3.714	1.939	4.154	1.573	2.769	1.787	3.077	1.935	3.846	1.625

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task4.TaskType (d)	0.009	0.009	0.003	0.002	0.006	0.006	0.007	0.01	0.007	0.007	0.006	0.008
Task4.TaskType (nf)	5.4	3.719	3.2	2.251	4.769	3.723	4.25	3.019	5.462	3.666	3.5	2.747
Task4.Question (d)	0.008	0.009	0.004	0.006	0.008	0.009	0.005	0.006	0.007	0.006	0.006	0.009
Task4.Question (nf)	4.733	3.218	5.0	4.574	5.308	2.81	4.5	4.604	4.786	4.406	4.933	3.432
Task5												
Task5.Bug1 (d)	0.019	0.02	0.009	0.005	0.015	0.014	0.014	0.017	0.02	0.021	0.01	0.008
Task5.Bug1 (nf)	32.667	22.983	34.571	22.954	35.769	24.563	31.812	21.476	32.308	17.09	34.625	26.743
Task5.Bug2 (d)	0.011	0.011	0.013	0.009	0.009	0.009	0.014	0.011	0.011	0.009	0.013	0.011
Task5.Bug2 (nf)	25.812	28.722	44.467	24.756	25.533	21.886	43.562	31.012	26.643	25.981	41.588	28.677
Task5.Line1 (d)	0.003	0.003	0.006	0.013	0.003	0.004	0.006	0.014	0.008	0.015	0.002	0.002
Task5.Line1 (nf)	6.556	5.151	22.182	48.4	8.0	8.667	22.3	50.715	26.0	56.775	7.917	8.051
Task5.Line2 (d)	0.004	0.004	0.003	0.003	0.004	0.004	0.002	0.002	0.004	0.004	0.003	0.003
Task5.Line2 (nf)	9.615	6.69	10.083	11.905	12.077	10.128	7.417	8.152	8.909	7.463	10.571	10.825
Task5.Line3 (d)	0.006	0.007	0.004	0.003	0.007	0.007	0.003	0.003	0.008	0.007	0.004	0.003
Task5.Line3 (nf)	14.643	13.101	13.615	10.882	18.462	12.914	10.143	9.574	15.818	11.566	13.0	12.302
Task5.Line4 (d)	0.004	0.003	0.004	0.003	0.003	0.003	0.004	0.004	0.004	0.003	0.003	0.004
Task5.Line4 (nf)	10.077	9.742	11.538	10.105	10.214	10.357	11.5	9.405	9.636	6.772	11.667	11.629
Task5.Line5 (d)	0.006	0.006	0.003	0.003	0.005	0.005	0.004	0.004	0.006	0.006	0.004	0.004
Task5.Line5 (nf)	14.062	15.212	11.643	10.463	13.467	15.165	12.4	11.051	11.571	9.205	14.125	15.891
Task5.Line6 (d)	0.009	0.008	0.004	0.003	0.007	0.008	0.006	0.005	0.009	0.007	0.005	0.005
Task5.Line6 (nf)	19.867	17.129	14.4	12.466	15.267	14.907	19.0	15.334	17.538	12.732	16.824	16.879
Task5.Line7 (d)	0.005	0.005	0.004	0.003	0.005	0.005	0.004	0.004	0.006	0.005	0.003	0.003
Task5.Line7 (nf)	10.308	10.451	11.583	8.393	11.25	10.402	10.615	8.675	11.818	8.388	10.214	10.289
Task5.Line8 (d)	0.008	0.008	0.005	0.004	0.008	0.008	0.006	0.004	0.008	0.008	0.006	0.005

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task5.Line8 (nf)	19.857	21.096	16.846	12.368	20.077	22.388	16.857	11.086	16.083	12.094	20.267	20.628
Task5.Line9 (d)	0.001	0.001	0.0	0.0	0.001	0.001	0.001	0.001	0.001	0.001	0.0	0.0
Task5.Line9 (nf)	1.571	0.787	1.4	0.548	1.714	0.756	1.2	0.447	1.5	0.535	1.5	1.0
Task5.Line10 (d)	0.006	0.003	0.004	0.003	0.006	0.003	0.005	0.003	0.006	0.004	0.005	0.003
Task5.Line10 (nf)	15.562	10.997	14.714	8.416	17.786	10.836	12.875	8.302	12.857	8.574	17.188	10.47
Task5.Line11 (d)	0.011	0.01	0.006	0.004	0.008	0.007	0.008	0.009	0.012	0.01	0.006	0.004
Task5.Line11 (nf)	21.75	12.969	21.533	13.032	22.867	13.902	20.5	11.978	22.286	12.35	21.118	13.481
Task5.Line12 (d)	0.015	0.018	0.008	0.008	0.011	0.012	0.012	0.018	0.017	0.019	0.008	0.008
Task5.Line12 (nf)	26.062	20.22	27.538	17.41	27.429	18.871	26.067	19.166	26.214	18.272	27.2	19.709
Task5.Line13 (d)	0.019	0.024	0.009	0.007	0.014	0.014	0.014	0.022	0.021	0.024	0.008	0.007
Task5.Line13 (nf)	31.75	27.733	31.0	19.537	34.857	29.276	28.375	18.344	34.214	25.595	28.938	22.773
Task5.Line14 (d)	0.015	0.016	0.009	0.007	0.014	0.016	0.01	0.009	0.017	0.016	0.009	0.008
Task5.Line14 (nf)	27.938	23.046	31.533	23.817	32.933	28.846	26.625	16.435	30.214	19.427	29.235	26.333
Task5.Line16 (d)	0.001	0.0	0.0	0.0	0.0	0.0	0.001	0.0	0.001	0.0	0.0	0.0
Task5.Line16 (nf)	1.167	0.408	2.0	1.414	1.333	0.816	1.8	1.304	1.0	0.0	2.0	1.265
Task5.Line17 (d)	0.008	0.009	0.009	0.005	0.006	0.006	0.01	0.007	0.007	0.005	0.01	0.008
Task5.Line17 (nf)	19.812	20.387	30.0	14.629	19.571	14.543	28.938	20.635	17.857	15.649	30.438	19.019
Task5.Line19 (d)	0.001	0.001	0.001	0.0	0.001	0.001	0.001	0.0	0.0	0.0	0.001	0.001
Task5.Line19 (nf)	2.5	3.0	1.833	1.602	2.4	2.608	1.8	1.789	2.0	2.0	2.167	2.401
Task5.Line20 (d)	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	nan	0.0	0.0
Task5.Line20 (nf)	1.333	0.577	1.0	0.0	1.5	0.707	1.0	0.0	1.0	nan	1.25	0.5
Task5.Line21 (d)	0.005	0.005	0.005	0.003	0.005	0.005	0.005	0.003	0.006	0.005	0.005	0.003
Task5.Line21 (nf)	13.933	11.889	21.929	11.29	16.462	11.163	18.875	13.068	14.857	10.458	20.533	13.212
Task5.Line22 (d)	0.008	0.005	0.007	0.006	0.007	0.006	0.007	0.005	0.008	0.006	0.006	0.005
Task5.Line22 (nf)	20.867	16.08	22.933	17.318	21.071	17.731	22.625	15.802	22.429	18.5	21.438	15.042

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task5.Line23 (d)	0.004	0.004	0.004	0.004	0.005	0.004	0.004	0.004	0.005	0.004	0.003	0.003
Task5.Line23 (nf)	10.286	9.547	12.923	8.026	11.917	8.404	11.267	9.354	12.154	10.723	11.0	6.884
Task5.Line24 (d)	0.005	0.004	0.004	0.003	0.004	0.004	0.004	0.004	0.004	0.003	0.004	0.004
Task5.Line24 (nf)	11.714	10.499	11.0	7.171	9.923	7.729	12.5	9.633	11.308	8.557	11.375	9.229
Task5.Line25 (d)	0.004	0.004	0.003	0.002	0.004	0.004	0.003	0.002	0.005	0.004	0.003	0.002
Task5.Line25 (nf)	10.0	6.563	11.154	5.475	10.917	7.025	10.267	5.23	10.538	6.553	10.571	5.639
Task5.Line26 (d)	0.0	0.0	0.001	0.001	0.0	0.0	0.001	0.001	0.001	0.001	0.0	0.0
Task5.Line26 (nf)	1.2	0.447	1.714	0.488	1.5	0.548	1.5	0.548	1.6	0.548	1.429	0.535
Task5.Line27 (d)	0.005	0.004	0.007	0.006	0.006	0.006	0.006	0.005	0.006	0.005	0.006	0.006
Task5.Line27 (nf)	15.286	12.443	24.133	17.121	19.615	18.635	20.062	12.948	18.615	14.327	20.875	16.701
Task5.Line28 (d)	0.009	0.009	0.01	0.008	0.009	0.01	0.009	0.008	0.011	0.01	0.007	0.007
Task5.Line28 (nf)	22.933	24.702	31.786	21.931	28.385	29.168	26.25	18.477	29.846	26.57	25.062	21.177
Task5.Line29 (d)	0.008	0.009	0.009	0.011	0.008	0.012	0.008	0.008	0.012	0.013	0.005	0.004
Task5.Line29 (nf)	19.333	18.63	28.533	30.507	25.214	32.13	22.812	18.367	30.571	34.389	18.125	11.529
Task5.Line30 (d)	0.007	0.007	0.006	0.007	0.007	0.008	0.006	0.006	0.008	0.009	0.005	0.005
Task5.Line30 (nf)	17.154	15.518	19.0	18.679	19.5	21.215	17.0	13.245	21.5	22.15	15.4	11.35
Task5.Line31 (d)	0.005	0.004	0.003	0.002	0.003	0.003	0.004	0.003	0.005	0.004	0.003	0.003
Task5.Line31 (nf)	13.071	10.752	10.714	7.353	10.667	8.627	12.812	9.642	12.75	10.217	11.25	8.489
Task5.Line32 (d)	0.001	0.0	0.001	0.0	0.001	0.0	0.0	0.0	0.001	0.0	0.001	0.0
Task5.Line32 (nf)	2.0	2.0	2.333	1.155	2.333	1.155	2.0	2.0	1.8	1.095	3.0	2.828
Task5.Line33 (d)	0.009	0.011	0.008	0.005	0.006	0.006	0.011	0.009	0.008	0.006	0.009	0.01
Task5.Line33 (nf)	25.286	26.682	29.067	20.475	18.714	18.053	35.2	25.386	25.25	23.16	28.647	24.039
Task5.Line34 (d)	0.011	0.008	0.009	0.006	0.007	0.006	0.012	0.007	0.011	0.008	0.009	0.007
Task5.Line34 (nf)	28.533	23.703	33.267	26.187	21.571	18.173	39.062	27.15	31.615	29.7	30.353	20.973
Task5.Line35 (d)	0.001	0.0	0.001	0.0	0.001	0.0	0.001	0.0	0.001	0.0	0.001	0.0

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task5.Line35 (nf)	2.0	0.707	2.2	0.837	2.25	0.957	2.0	0.632	2.0	0.707	2.2	0.837
Task5.Line36 (d)	0.005	0.004	0.006	0.003	0.005	0.004	0.006	0.003	0.006	0.004	0.005	0.003
Task5.Line36 (nf)	15.4	11.909	20.143	15.357	14.538	12.514	20.25	14.378	16.462	12.791	18.688	14.641
Task5.Line37 (d)	0.006	0.004	0.004	0.003	0.003	0.004	0.006	0.004	0.006	0.004	0.004	0.003
Task5.Line37 (nf)	17.308	14.739	13.5	12.315	11.083	11.277	18.733	14.36	15.083	13.648	15.533	13.695
Task5.Line38 (d)	0.0	0.0	0.001	0.0	0.0	0.0	0.001	0.0	0.001	nan	0.0	0.0
Task5.Line38 (nf)	1.5	0.707	1.5	0.707	1.5	0.707	1.5	0.707	2.0	nan	1.333	0.577
Task5.Line39 (d)	0.003	0.003	0.003	0.002	0.003	0.002	0.004	0.003	0.003	0.001	0.004	0.003
Task5.Line39 (nf)	9.769	11.541	10.0	6.232	8.167	7.146	11.357	10.515	7.0	5.354	12.769	11.196
Task5.Line40 (d)	0.005	0.005	0.003	0.003	0.003	0.003	0.006	0.004	0.003	0.002	0.005	0.005
Task5.Line40 (nf)	15.846	16.03	10.615	6.862	8.714	8.489	18.5	14.369	9.5	6.842	16.429	15.195
Task5.Line41 (d)	0.005	0.006	0.005	0.004	0.004	0.006	0.006	0.005	0.005	0.003	0.006	0.007
Task5.Line41 (nf)	13.933	15.088	18.077	13.895	13.571	14.727	18.143	14.298	13.692	13.732	17.733	15.229
Task5.Line42 (d)	0.007	0.007	0.008	0.009	0.006	0.007	0.009	0.009	0.008	0.006	0.007	0.009
Task5.Line42 (nf)	17.5	16.769	26.643	23.405	18.643	16.864	25.5	23.754	23.75	22.499	20.812	19.552
Task5.Line43 (d)	0.009	0.008	0.008	0.005	0.007	0.005	0.011	0.008	0.01	0.008	0.008	0.005
Task5.Line43 (nf)	22.333	18.627	29.308	18.468	19.571	13.976	31.571	21.015	27.077	23.06	24.267	14.275
Task5.Line44 (d)	0.007	0.006	0.006	0.004	0.006	0.005	0.006	0.005	0.007	0.004	0.006	0.006
Task5.Line44 (nf)	19.308	17.216	21.071	15.701	19.818	17.865	20.5	15.466	18.833	10.504	21.333	19.87
Task5.Line45 (d)	0.006	0.005	0.007	0.005	0.005	0.005	0.007	0.004	0.006	0.004	0.006	0.006
Task5.Line45 (nf)	16.5	18.55	22.231	16.893	17.0	19.79	21.067	16.259	15.846	9.72	22.429	22.691
Task5.Line46 (d)	0.003	0.002	0.005	0.004	0.003	0.002	0.004	0.004	0.004	0.003	0.004	0.004
Task5.Line46 (nf)	7.75	6.398	15.0	13.87	8.7	6.584	13.286	13.499	9.364	6.185	13.077	14.192
Task5.Line47 (d)	0.001	0.001	0.002	0.002	0.001	0.001	0.001	0.002	0.001	0.001	0.002	0.002
Task5.Line47 (nf)	4.0	3.899	6.444	7.78	4.111	2.934	5.909	7.622	3.0	2.404	7.2	7.642

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task5.Line48 (d)	0.0	0.0	0.0	nan	0.0	0.0	n/a	n/a	0.0	nan	0.0	0.0
Task5.Line48 (nf)	1.0	0.0	1.0	nan	1.0	0.0	nan	nan	1.0	nan	1.0	0.0
Task5.Line49 (d)	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a	n/a
Task5.Line49 (nf)	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
Task5.Code (d)	0.131	0.063	0.102	0.052	0.102	0.064	0.132	0.051	0.126	0.05	0.11	0.065
Task5.Code (nf)	331.375	232.674	366.467	190.858	297.133	196.253	396.375	218.496	308.5	198.416	381.176	220.684
Task5.Help (d)	0.001	0.001	0.113	0.091	0.058	0.055	0.131	0.106	0.129	0.124	0.081	0.071
Task5.Help (nf)	1.5	0.707	284.143	183.136	172.143	149.173	308.444	214.808	295.167	260.61	221.0	154.773
Task5.HelpButton (d)	0.002	0.003	0.001	0.001	0.002	0.003	0.002	0.002	0.002	0.003	0.001	0.001
Task5.HelpButton (nf)	2.867	2.066	4.385	2.468	3.357	2.098	3.786	2.636	3.571	2.344	3.571	2.441
Task5.TaskType (d)	0.003	0.002	0.001	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002	0.002
Task5.TaskType (nf)	4.636	3.443	4.364	6.104	4.4	2.951	4.583	6.127	4.417	5.961	4.6	3.34
Task5.Question (d)	0.008	0.012	0.008	0.006	0.006	0.006	0.01	0.012	0.013	0.012	0.004	0.004
Task5.Question (nf)	12.75	15.507	21.333	14.998	16.4	17.683	17.375	14.004	23.929	18.074	11.118	10.665
Task6												
Task6.Line1 (d)	0.001	0.0	0.001	0.001	0.001	0.001	0.001	0.0	0.0	0.0	0.001	0.001
Task6.Line1 (nf)	2.0	1.528	3.429	3.409	3.125	3.137	2.167	1.941	1.5	0.837	3.625	3.204
Task6.Line2 (d)	0.002	0.002	0.003	0.002	0.003	0.002	0.002	0.002	0.002	0.002	0.003	0.002
Task6.Line2 (nf)	4.2	3.084	8.444	6.366	5.818	4.792	6.75	6.112	4.667	2.784	7.6	6.603
Task6.Line3 (d)	0.002	0.003	0.005	0.005	0.004	0.004	0.003	0.004	0.002	0.003	0.005	0.005
Task6.Line3 (nf)	4.727	6.436	12.818	11.881	9.7	10.894	8.0	10.009	4.5	5.729	13.9	12.215
Task6.Line4 (d)	0.001	0.001	0.001	0.0	0.001	0.001	0.001	0.0	0.001	0.0	0.001	0.0
Task6.Line4 (nf)	1.833	1.169	2.2	0.837	2.167	1.169	1.8	0.837	1.667	0.577	2.125	1.126
Task6.Line5 (d)	0.001	0.0	0.0	0.0	0.001	0.0	0.0	0.0	0.0	0.0	0.001	0.0

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task6.Line5 (nf)	1.0	0.0	1.25	0.5	1.333	0.577	1.0	0.0	1.0	0.0	1.25	0.5
Task6.Line6 (d)	0.006	0.006	0.005	0.006	0.005	0.006	0.005	0.006	0.006	0.006	0.004	0.005
Task6.Line6 (nf)	9.2	7.984	10.5	9.929	10.143	11.002	9.533	6.599	9.923	7.577	9.75	9.99
Task6.Line7 (d)	0.005	0.006	0.004	0.002	0.004	0.003	0.005	0.006	0.003	0.002	0.006	0.006
Task6.Line7 (nf)	8.0	6.114	11.0	5.878	9.273	5.442	9.467	6.696	7.385	4.073	11.385	7.194
Task6.Line8 (d)	0.005	0.003	0.004	0.004	0.005	0.004	0.004	0.003	0.004	0.003	0.005	0.004
Task6.Line8 (nf)	8.923	6.157	10.308	8.606	9.667	7.912	9.571	7.165	7.833	5.078	11.143	8.778
Task6.Line9 (d)	0.001	0.0	0.001	0.0	0.001	0.0	0.001	0.0	0.0	0.0	0.001	0.0
Task6.Line9 (nf)	2.0	1.265	2.0	1.0	2.0	1.0	2.0	1.265	1.0	0.0	2.375	1.061
Task6.Line10 (d)	0.001	0.0	n/a	n/a	n/a	n/a	0.001	0.0	0.001	nan	0.001	nan
Task6.Line10 (nf)	1.5	0.707	nan	nan	nan	nan	1.5	0.707	2.0	nan	1.0	nan
Task6.Line11 (d)	0.004	0.005	0.003	0.002	0.002	0.002	0.004	0.005	0.005	0.005	0.002	0.002
Task6.Line11 (nf)	6.231	5.372	8.0	4.632	4.727	2.76	8.929	5.663	8.667	4.774	5.615	4.942
Task6.Line12 (d)	0.003	0.003	0.002	0.001	0.002	0.001	0.002	0.003	0.003	0.003	0.002	0.001
Task6.Line12 (nf)	5.7	4.832	4.545	2.583	4.5	2.268	5.462	4.502	6.1	4.067	4.182	3.401
Task6.Line13 (d)	0.005	0.005	0.006	0.003	0.004	0.003	0.007	0.005	0.006	0.005	0.005	0.004
Task6.Line13 (nf)	10.429	9.565	16.0	8.388	8.727	3.552	16.133	10.986	13.417	8.795	12.643	10.035
Task6.Line14 (d)	0.007	0.004	0.008	0.005	0.005	0.004	0.01	0.004	0.009	0.004	0.006	0.004
Task6.Line14 (nf)	14.0	11.225	19.154	8.877	10.077	6.946	22.429	9.428	19.417	10.833	14.133	9.583
Task6.Line15 (d)	0.002	0.001	0.0	0.0	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Task6.Line15 (nf)	2.625	1.302	1.625	0.744	2.2	1.304	2.091	1.136	1.875	1.356	2.375	0.916
Task6.Line16 (d)	0.012	0.009	0.012	0.01	0.012	0.011	0.012	0.008	0.014	0.011	0.01	0.007
Task6.Line16 (nf)	21.8	17.346	31.214	19.192	23.929	18.809	28.6	18.681	27.571	19.622	25.2	18.119
Task6.Line17 (d)	0.033	0.061	0.022	0.019	0.033	0.062	0.022	0.019	0.036	0.063	0.019	0.015
Task6.Line17 (nf)	33.75	35.271	52.0	38.563	34.6	32.311	49.933	41.491	39.0	36.428	45.533	39.263

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task6.Line18 (d)	0.001	0.001	0.0	0.0	0.0	0.0	0.001	0.001	0.0	0.0	0.001	0.001
Task6.Line18 (nf)	2.4	1.14	1.778	1.093	1.286	0.488	2.714	1.113	1.714	0.756	2.286	1.38
Task6.Line19 (d)	0.001	0.001	0.0	nan	0.0	0.0	0.001	0.001	0.0	nan	0.001	0.001
Task6.Line19 (nf)	1.75	1.5	1.0	nan	1.0	0.0	2.0	1.732	1.0	nan	1.75	1.5
Task6.Line20 (d)	0.006	0.006	0.004	0.006	0.006	0.008	0.005	0.004	0.005	0.006	0.006	0.007
Task6.Line20 (nf)	11.0	9.577	11.923	8.732	9.692	9.621	12.933	8.54	10.692	8.509	12.067	9.721
Task6.Line21 (d)	0.008	0.008	0.008	0.007	0.008	0.008	0.008	0.007	0.008	0.008	0.008	0.007
Task6.Line21 (nf)	15.857	12.309	21.769	16.382	14.417	10.578	22.133	16.47	17.692	14.424	19.643	14.944
Task6.Line22 (d)	0.009	0.009	0.01	0.007	0.008	0.01	0.011	0.006	0.011	0.009	0.008	0.006
Task6.Line22 (nf)	17.0	13.737	26.5	19.174	15.929	13.35	26.867	18.692	22.214	15.9	21.0	18.474
Task6.Line23 (d)	0.009	0.007	0.014	0.008	0.01	0.008	0.012	0.008	0.014	0.009	0.009	0.007
Task6.Line23 (nf)	17.867	16.788	34.5	19.234	20.154	16.502	30.562	21.156	28.615	18.333	23.688	20.899
Task6.Line24 (d)	0.001	0.001	0.001	0.002	0.001	0.001	0.001	0.001	0.001	0.002	0.001	0.001
Task6.Line24 (nf)	2.25	1.035	2.818	2.442	2.0	1.512	3.0	2.191	3.111	2.522	2.1	1.197
Task6.Line25 (d)	0.01	0.005	0.013	0.012	0.011	0.013	0.012	0.006	0.014	0.012	0.01	0.006
Task6.Line25 (nf)	17.286	6.73	28.333	21.979	19.385	14.517	25.938	18.972	24.429	18.867	21.667	15.931
Task6.Line26 (d)	0.011	0.007	0.025	0.019	0.015	0.015	0.02	0.016	0.019	0.016	0.017	0.016
Task6.Line26 (nf)	21.2	14.891	54.214	43.336	29.538	25.32	43.312	41.919	38.769	30.169	35.812	40.354
Task6.Line27 (d)	0.014	0.01	0.016	0.015	0.011	0.008	0.018	0.015	0.02	0.016	0.01	0.006
Task6.Line27 (nf)	29.733	32.345	39.2	34.865	24.214	15.333	43.438	42.049	44.714	41.779	25.5	21.432
Task6.Line28 (d)	0.0	0.0	0.001	0.001	0.0	0.0	0.001	0.001	0.001	0.001	0.0	0.0
Task6.Line28 (nf)	1.167	0.408	2.8	1.643	1.286	0.488	3.0	1.826	1.8	1.304	2.0	1.549
Task6.Line29 (d)	0.011	0.009	0.009	0.009	0.01	0.01	0.011	0.007	0.012	0.009	0.009	0.008
Task6.Line29 (nf)	20.0	15.128	23.333	18.829	16.786	11.833	25.938	19.672	24.143	18.708	19.5	15.362
Task6.Line30 (d)	0.007	0.004	0.007	0.004	0.006	0.004	0.008	0.004	0.008	0.004	0.006	0.004

Continued on next page

Table F.9 – continued from previous page

Variable	**Condition**				**Expertise**				**Performance**			
	Control		**Help**		**Low**		**High**		**Low**		**High**	
	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ	$\overline{x}$	σ
Task6.Line30 (nf)	13.143	7.502	19.4	13.255	11.462	6.875	20.375	12.484	17.857	11.072	15.0	11.408
Task6.Line31 (d)	0.006	0.005	0.008	0.007	0.005	0.005	0.008	0.007	0.008	0.006	0.006	0.006
Task6.Line31 (nf)	11.467	10.467	17.154	10.082	9.538	6.527	18.067	11.835	15.0	11.292	13.333	10.104
Task6.Line32 (d)	0.001	nan	0.001	0.001	0.001	nan	0.001	0.0	0.001	nan	0.001	0.0
Task6.Line32 (nf)	3.0	nan	2.0	1.414	1.0	nan	3.0	0.0	3.0	nan	2.0	1.414
Task6.Line33 (d)	0.001	nan	0.002	nan	n/a	n/a	0.001	0.0	0.002	nan	0.001	nan
Task6.Line33 (nf)	3.0	nan	2.0	nan	nan	nan	2.5	0.707	2.0	nan	3.0	nan
Task6.Code (d)	0.142	0.085	0.125	0.075	0.107	0.082	0.158	0.071	0.139	0.071	0.128	0.089
Task6.Code (nf)	264.188	193.346	322.933	199.12	211.8	139.601	368.375	212.902	295.667	226.109	289.75	168.625
Task6.Help (d)	0.002	0.003	0.078	0.08	0.038	0.054	0.089	0.09	0.069	0.102	0.066	0.053
Task6.Help (nf)	1.5	0.707	180.75	167.965	101.333	147.963	195.5	179.502	120.714	134.008	189.571	200.446
Task6.HelpButton (d)	0.011	0.034	0.002	0.002	0.011	0.033	0.002	0.002	0.011	0.033	0.002	0.002
Task6.HelpButton (nf)	3.0	2.174	5.571	4.256	3.923	2.813	4.846	4.375	3.846	2.193	4.923	4.699
Task6.TaskType (d)	0.004	0.003	0.001	0.001	0.003	0.003	0.002	0.002	0.003	0.003	0.002	0.002
Task6.TaskType (nf)	6.231	5.118	3.75	2.896	4.583	4.87	5.462	3.865	5.308	5.376	4.75	2.958
Task6.Question (d)	0.007	0.013	0.003	0.002	0.004	0.009	0.006	0.012	0.009	0.014	0.002	0.002
Task6.Question (nf)	14.133	30.692	8.455	7.313	7.0	7.495	15.786	31.391	17.167	33.989	7.071	6.569

Sander Bjerklund Lindberg

Exposing novice programmers to an expert's eye-gaze

NTNU
Norwegian University of
Science and Technology