Gaze Insights into Debugging Behavior Using Learner-Centred Analysis

Katerina Mangaroska
Norwegian University of Science and Technology
Trondheim, Norway
mangaroska@ntnu.no

Kshitij Sharma
Norwegian University of Science and Technology
Trondheim, Norway
kshitij.sharma@ntnu.no

Michail Giannakos
Norwegian University of Science and Technology
Trondheim, Norway
michailg@ntnu.no

Hallvard Trætteberg
Norwegian University of Science and Technology
Trondheim, Norway
hal@ntnu.no

Pierre Dillenbourg
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
pierre.dillenbourg@epfl.ch

ABSTRACT

The presented study tries to tackle an intriguing question of how user-generated data from current technologies can be used to reinforce learners' reflections, improve teaching practices, and close the learning analytics loop. In particular, the aim of the study is to utilize users' gaze to examine the role of a mirroring tool (i.e. Exercise View in Eclipse) in orchestrating basic behavioral regulation of participants engaged in a debugging task. The results demonstrated that students who processed the information presented in the Exercise View and acted upon it, improved their performance and achieved higher level of success than those who failed to do it. The findings shed a light how to capture what constitute relevant data within a particular context using gaze patterns, that could guide collection of essential learner-centred analytics for the purpose of designing usable and modular learning environments based on data-driven approaches.

CCS CONCEPTS

• **Human-centered computing → Empirical studies in HCI**; **User centered design**; • **Social and professional topics → Computer science education**;

KEYWORDS

Eye-tracking, Learner-centred analysis, Mirroring tools, Behaviour regulation, Debugging

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

To gain greater understanding in user's needs and behaviors, researchers should utilize the available multifaceted user-generated data, that will empower them to design technology enhanced human learning. More specifically, they should employ various modalities of user-centred analytics to improve learning, predict behavior, and advice users and educators by converting educational data into meaningful information. However, most of the educational systems still have concerns regarding what constitutes relevant data [38], and fail to consider user's perspective, as well as actively involve users in the design and development of learning experiences. Moreover, majority of past studies [25] focus on data extraction based on availability (e.g. click-stream data from learning management systems), rather than on data that cannot be captured verbally, with learning systems or observations. Hence, this study aims to use eye-tracking to gain insights into user debugging behavior (e.g. how user process information or interact with visual information) utilizing a mirroring tool. The mirroring tool is a special plug-in (i.e. Exercise View in Eclipse) that captures user programming actions and visualizes them to the users. The idea behind the mirroring tool is to raise awareness of user's own actions and reinforce reflective learning. Through this paper we contribute to the knowledge base of learning-centred analysis by providing evidence how mirroring tools can orchestrate behavior regulation skills within academic context.

On the other hand, gaze data have already proven its value in understanding how students learn to program and debug [34]. However, despite the great potential, little consideration has been given to how gaze data can be combined with other forms of more conventional learning analytics (e.g. click-stream) to inform user-centred design. Taking this into consideration, the aim of the study is to examine the role of Exercise View into user's debugging behavior and empower multimodal user-centred analysis to design relevant learning strategies for programming/debugging. As a result, this study addresses the following research questions:

Question 1: What is the level of students' visual attention to the mirroring tool (i.e. Exercise View)?

Question 2: How students' expertise, success, and gaze patterns are associated?

Question 3: How does the time spent on the mirroring tool relate to the performance (i.e. code produced to solve a task)?
Question 4: What gaze patterns relate to high debugging success and what is the role of mirroring capabilities (i.e. Exercise view)?

2 RELATED WORK

2.1 User-centred design and learning analytics

Digital technology is rapidly changing the way students learn and engage in learning activities. At the same time huge amount of user-generated data is becoming available. In order to utilize the produced data streams to support the design of user-centred learning systems, new tools and practises are required. To accomplish this requirement and design user-centred learning environments that are usable and underpin a robust learning experiences, there is an ongoing challenge to converge techniques and methods from interdisciplinary fields such as software engineering, cognitive sciences, and technology enhanced learning (TEL) [2]. To cope up with this goal, researchers need to utilize learning analytics to inform learning designs, and develop data-driven tools and learning strategies that are appropriate to enhance teaching practices and student learning experiences [24, 25, 35].

Most of the current educational systems focus on educators' view and rarely involve students in the design and development processes. For instance, a literature study [13] documented that from 22 learning analytics related systems only 5 are designed purely to be used by students, and only few are available for general use. Martinez-Maldonado et al. [26] employed a five-stage LATUX work flow to design, deploy, and validate awareness tools in TEL, utilizing user-centred analysis. The study underlined the importance of user-centred design as a process of refining and designing visual awareness tools for specific contexts. Moreover, Wise [40] proposed a pedagogical intervention design concept that emphasize the importance of enhancing existing learning analytics practices in the design of user-centred teaching and learning activities considering new technologies. Dawkins [12] working at the intersection of learning analytics and user-centred design, depicted the significance of 'content strategy' as a valuable and missing component of learning analytics for improving learning design. This aspect has been neglected in many studies as data extraction is based on availability rather than users needs. Thus, selecting and deciding how to employ different learning analytics to support user-centred design is a challenging task and requires contextual information for interpretation. Consequently, one of the aims in this study is to examine the contextual set up of a programming environment for the purpose of observing what constitutes relevant data when learners engage in problem-solving tasks (i.e. debugging task), as this information could be a valuable input in designing user-centred learning experiences for teaching and learning programming using gaze data.

2.2 Eye-tracking and user-centred design

Eye-tracking helps researchers to gain insights into user behavior (e.g. how the user process information or interacts with visual information) that cannot be captured verbally, via other more ordinal user-data (e.g. click-stream) or with observations [11]. Moreover, in educational technology, designers and researchers need to design systems with high-quality user experience and high degree of learnability (i.e. learning experience) [16]. In order to accomplish this, the systems need to follow user-centred design guidelines, maintain minimal errors, and avoid user-frustration [8].

In eye-tracking studies, saccades and fixations are the main indicators of user behavior for interacting and processing information. Previous studies [1, 15, 17, 37] used eye-tracking data to evaluate and improve the design and the user experience in different development stages. In one of the studies [15] the authors reported that high number of saccades indicates long time to search for information, while a high number of fixations represents a high degree of user uncertainty. In a different study [28], researchers evaluated the design of a Learning Management System (LMS) interface and highlighted the user's need for simplicity, memorability (e.g. how easy is it for the returning user to perform effectively) and control over the interaction with the LMS. The findings from these studies support Federici's idea that eye-tracking is a "psychotechnology" because it emphasizes the intra-systemic perspective of the relation between the user and the technology [27]. These studies show that eye-tracking provides a unique access to user's behavior (in educational contexts, learners), hence one can use the learners' gaze data to extract information and feed the data back to the learners while efficiently closing the LAK loop [10]. Thus, as eye-tracking is considered to be an objective technique that can provide insights into many aspects of human cognitive abilities, such as problem solving, reasoning, and mental strategies [3, 22, 41], gaze data can be utilized to inform the design of learning technologies by effectively covering wide range of user's needs and behaviors.

As it can be seen from the past studies [14, 27], eye-tracking is a promising methodology for collecting data, measuring user behavior, and the amount of processing (e.g. ease of use, perceived playfulness, cognitive load) when users interact with computer systems. This is a relevant way of observing the dynamic trace *where is the user's attention directed in relation to the presented visual information*, so that designers can embrace and practice user-centred design.

2.3 Eye-tracking and problem solving

Most results show that eye-tracking allows researchers to get attentional data for users while they perform tasks. Likewise, previous studies [19, 21, 23, 29] had shown that eye-tracking can explain various constructs like contextual expertise, task dependency, and task complexity.

Reingold et. al, [29] conducted a study to understand how the expertise of chess players (novices, intermediate, and experts) and their way of encoding a given chessboard state are related. Using two different chess configurations (random and original) the researches tried to calculate the area of visual span while participants were asked to detect a modified piece. The results showed that experts had larger area of visual span and were faster in detecting modifications in original game configurations than in random, while there were no differences found among novices and intermediate players. Moreover, the researchers reported that experts do encode larger chunk of the configuration from the rest of the players due to their use of foveal and parafoveal regions [9].

Considering the relation between task dependency and gaze patterns, Kaller et al. [23] conducted a study to gain a better understanding of the time course of visuospatial problem solving. During the experiment, the participants were randomly assigned to two groups depending on where the start state and the goal state were presented. For the start-goal group, the start states were always presented at the left side and the goal states were presented on the right side. For the goal-start group, the arrangement was inverted. During the initial thinking time [1] most of the subjects directed their fixations mainly to the left side irrespective of the state arrangement. As a result, the authors discovered a strong dependency between the personal preferences and the gaze patterns. Moreover, the data from the experiment displayed task-dependent eye-movement patterns with respect to the subsequent gaze alternations, supporting a sequential model of problem solving as internalization, planning, execution, and verification.

Another relevant study was carried out comparing car drivers driving while performing (or not performing) arithmetic tasks [19]. The results showed that drivers pay less attention to the mirrors, instruments, and the peripherals while performing a task rather than when they have no additional cognitive task other than focusing on driving. Moreover, the subjective ratings about the cognitive load, reduction of safety, and distraction was found to be increased from no task to easy task to difficult task conditions.

Jones [21] used a Car Park problem[2] to find the relation between the problem solving processes and the gaze data. The authors looked at the fixation time three moves prior to the object car move and three moves after the object car move. The fixation time on the problem was longer for the object car move, than that in the prior or succeeding moves to the object car move. Moreover, non-solvers spent significantly more time on the free area than the solvers.

Aforementioned studies show that a relation exists between the gaze patterns and the expertise (e.g. chess players), task complexity (e.g. driving), and the task-based performance (e.g. car park problem). Hence, in the next subsection the authors give examples of a specific problem solving tasks (i.e. debugging a code) which were considered while setting up the research task in this study.

2.4 Eye-tracking and debugging

Previous eye-tracking studies [4–7] show a clear relation between gaze patterns and task-based performance in debugging. Bednarik et al. [6] conducted a study of restricted focus viewer (RFV) [3] against a control condition (without RFV) in a debugging task. The results showed no significant difference between the two experimental conditions, but displayed a relation between debugging success and expertise. In terms of gaze behavior, the RFV condition induced more switches from the code area to the output area than the control condition [6]. In a different study, Bednarik et al. [7] investigated the relation between expertise, gaze among code visualization, and debugging success of programmers. There was a relation between expertise and success, but the results were non-conclusive regarding the use of the visualization in the debugging task. Finally, in a most recent study, Bednarik [4] investigated whether programmers who did well (regardless of expertise) were also the ones who used and integrated different information sources more than the programmers who did not perform well. The author found that greater expertise allowed participants to spend more time integrating the information from multiple representations, but the difference was more significant during the last stages of the debugging task.

Moreover, in [32] authors compared the first scan time [4] against the different levels of debugging success. The results showed that successful debuggers had a significantly lower first scan time than the less successful ones. In terms of gaze behavior, this study showed that successful debuggers had a more vertical gaze than those who perform less successfully during the debugging task.

3 RESEARCH OBJECTIVE

The presented examples have shown that visual attention could be an important proxy for understanding the mechanisms underlying learning to program or to debug. However, the current understanding of the role of visual attention in program comprehension or debugging and coordination of representations is still in an early stage. Previous studies [4, 6, 7, 36] have tried to use eye-tracking to capture programmers' visual attention while debugging (i.e. finding and reporting errors/bugs) and construct interpretations of individual differences and expertise. Nonetheless, it has not been sufficiently explored whether and how visual attention strategies contribute in the process of learning programming and debugging, and how the strategies differ across various levels of expertise. Consequently, this study attempts to explore this issue further, by adding novelty and uniqueness to the research design of the performed experiment. The following are the salient characteristics of the present study:

(1) ***We allowed participants to edit the code***. This way we have more complete task than the studies described in Section 2.4. Since the debugging process consists of three phases: program comprehension, finding the bug, and removing the bug, not allowing students to write the code, makes us fail in capturing the last phase of the debugging process. Therefore, instead of just finding the bugs as it was the case in [4, 6, 7, 33], the participants in the present study were also required to remove those bugs.

(2) ***We have a more systematic way of debugging in a form of unit tests***. We designed the unit tests for the code and the participants were required to solve all of them in a designated time period. This was not the case with the experiments reported in Section 2.4. Moreover, in most of the studies the participants were provided with a sample code and one instance of the output [4, 6, 30, 31], and they could not edit the code. As we pointed in the previous paragraph, not letting the participants to debug the program, hinders the understanding of the debugging process.

(3) In the present study, ***we designed a mirroring tool (as a plug-in to Eclipse), that reflects the progress to the participants, as well as their successes or failures during the debugging task***. The successes or failures were based on the number of unit tests passed by each participant.

[1] time between the presentation of the problem and onset of the first action

[2] The goal of the car park problem is to manoeuvre a car out of a parking space. The parking space has other cars as well, which can be moved only in their initial orientation.

[3] an RFV is a special tool that blurs parts of the screen to make participants focus on a specific AOI.

[4] the time takes by the participants to read the code for the first time.

(4) Previous studies [6, 30] used an external graphical representations of the code (data flow, control flow, class-object diagrams) to aid the understanding of the participants. However, these representations might hinder the understanding of the code if the participants had no previous knowledge of using representations in learning programming; as it was evident from the reported gaze patterns. The participants did not pay much attention to the visual representations. In our study, ***we used the "variable view" provided by the Eclipse IDE to show the state of different selected variables*** during the debugging of the code. Participants could choose when to enable this view and when to remove this view from the screen if they find it redundant.
(5) In terms of ***eye-tracking analysis***, the saliency comes from the previous four points. Since we allowed the participants to edit the code and execute it as many times as they want it, there were frequent switches among the code, errors, and output areas of the IDE. Moreover, frequent switches were observed between the code and the unit test panels. When the participants enabled the "variable view" to get the help from the IDE to debug the code, this action added an Area of Interest (AOI) on the screen. Another important difference was the number of used AOIs. In most of the reported studies, the authors used three AOIs (code, visualization, output) and few of them used a fourth one, called sequence. In our study we decided to define 7 main AOIs (i.e. code, exercise view, JUnit test, problems, console, variable view, and debug view) to get a better understanding of users cognitive skills. Additionally for completion we added two more AOIs based on the IDE: toolbar and project explorer. Table 1 presents the detailed description of the AOIs. Finally, during the analysis stage, we decided to calculate two and three-way transitions for greater understanding of user's cognitive needs, as well as higher validity and reliability of the study, while previous studies reported in Section 2.4 analyzed only two-way transitions.

Table 1: Descriptions of the AOIs in Eclipse IDE.

AOI	Description
Toolbar	the toolbar of the IDE.
VariableView (VV)	during a debug, allows to change the value of a variable to test how your program handles a particular value or to speed through a loop.
DebugView (DV)	allows to manage the debugging or running of a program in the workbench.
ProjectExplorer	provides a hierarchical view of the artifacts in the workbench.
JUnit (JU)	allows to list the unit tests to be passed by the main Java class.
Code	panel where the code is written.
ExerciseView (EV)	allows to see the coding, saving, testing, and progress.
Problem (P)	shows the errors and/or warnings raised by the Java Compiler.
Console (C)	shows the output of the code.

4 METHODOLOGY

4.1 Debugging activity

The authors designed and implemented a debugging activity in conjunction with the partners from the University École Polytechnique Fédérale de Lausanne. The main task assigned to the participants was debugging a Java class named Person, that manages parent-child relationships. The provided code tried, but failed to ensure consistent object relationships, such as a mother of a child is female and a father of a child has that child in its list of children. The participants could check the correctness of the code by running the provided unit tests.

4.2 Participants

During the spring 2017, an experiment was performed at a contrived computer lab setting at the University École Polytechnique Fédérale de Lausanne with 40 computer science majors (12 females and 28 males) in their third semester. The mean age of the participants was 19.5 years (Std. Dev. = 1.65 years). In the previous semester, all of the participants had taken a Java course, where they were predominantly using Eclipse as Integrated Development Environment (IDE). Moreover, they were also familiar with the built-in debugging tool provided by Eclipse. The focus of this study is examining how user-generated gaze data can be used to reinforce students' reflection practices. Moreover, the study also considered whether students can practice problem-solving strategies (e.g. debugging a code) coupled with reflections (from a mirroring tool) rather than trial and error attitude.

4.3 Procedure

Upon arrival in the laboratory, the participants signed an informed consent form. After this and prior to the debugging task, each participant had to pass an automatic eye-tracking calibration routine to accommodate the eye tracker's parameters to each participant eyes to ensure accuracy in tracking the gaze. Their gaze during the debugging task was recorded using an SMI RED 250 eye-tracker at 250Hz. Next, the participants were asked to perform a pre-task, which required removing 90 errors from a skeleton code within 10 minutes. After this task, the participants were given 40 minutes to solve 5 debugging tasks presented as a part of the main method of the main class of a 100 lines of Java code. The code for the main debugging task contained no syntactic errors, and the participants were notified about this fact. Throughout the experiment we recorded and later analyzed participants' fixations (i.e. the state when the eye remains still over a period of time) and saccades (i.e. the rapid motion of the eye from one fixation to another) [20]. For their participation in the experiment, the participants were rewarded with an equivalent of CHF 25.

4.4 The mirroring tool

For the purpose of the experiment, the students were using the Eclipse IDE to complete the exercise. The exercise had pre-written unit tests and 5 questions that led the students to check the correctness of the code and debug it. Their Eclipse installation had been extended with an Exercise View (EV) plug-in that collected data from their use of Eclipse (see Figure 1). The data that this plug-in

collected and mirrored it back to the students included: lines of code, number of errors and warnings in the code, how many times the standard Java main method was launched, the unit test results (success, failure or error), debugging events (e.g. stopping on break points or resuming execution), and execution of commands (e.g. stepping through code). The success, failure or error of the tests give students some type of feedback about their progress that could support them to incrementally work towards the exercise's learning goals. It is more than obvious that students could not learn to debug in 40 minutes, but researchers could observe the gaze transitions between the different elements in the IDE, as well as when and how often students attend to the information that EV reflects. This information could later be used to implement user-centred approach in designing learning strategies for programming and debugging.

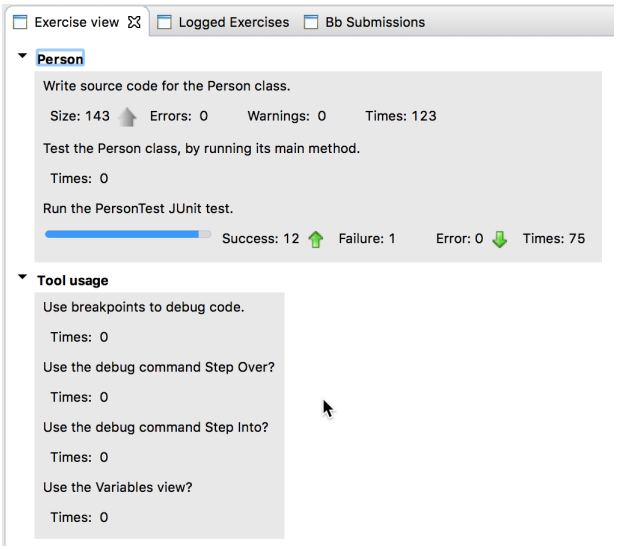

Figure 1: The Exercise View configured for the experiment. The top panel shows the progress of the "Person" class. The first line shows lines of code, number of errors and warnings by the Java compiler, and the number of times the code has been compiled. The second line shows the number of times the code has been run (not as part of a unit test). The third line shows the counts of unit tests. Arrows show the change of the metric, the direction shows the increase/decrease of the metric, while the color shows whether the change is considered as an improvement (i.e. green) or not (i.e. red). The bottom panel shows statistics from using the Eclipse debugger.

4.5 Variables

Expertise – The expertise was decided by the pre-task test, where the students were presented with a task of removing 90 errors from a skeleton code. The simple instruction was to write the minimal "stub" for a Java class so that the errors could be removed. The students were given 10 minutes to complete the pre-task. All but 5 students were able to finish the pre-task in the allotted amount of time. These five students were labelled as "novices". The rest of the 35 students were labelled as "experts" or "novices" based on the minimalism in their code.

Debugging success – For the debugging task, there were 10 unit tests prepared by the instructor (see subsection Procedure). To limit the debugging to one of the panels of the Eclipse IDE, the researchers introduced few bugs in otherwise complete code that would make the code fail all 10 unit tests. In order to pass all of the unit tests, the students were required to solve the debugging exercises in a particular order. Participants were given 40 minutes to complete the task. At the end of the 40 minutes, they were told to stop, and the number of unit tests passed at that point of time was taken to be the measure of the "debugging success".

Individual Areas of Interest (AOIs) – Eclipse IDE was divided into nine AOIs. Table 1 gives a brief overview of all AOIs for this experiment. During the analysis, the researchers computed the proportion of time spent on each of the AOIs as well as the transitions between the different AOIs. The results for the AOIs were later compared to the participant performance and expertise.

Transitions among AOIs – For the purpose of data analysis, the researchers decided to compute the transition probability of moving from one AOI to another. This is called a two-way transition probability. This type of transition shows the attention shift from one part of IDE to another, which might correspond to a specific behavioral pattern while students were debugging the code. For example, the transition from code to console might depict the behavior of having a hypothesis about the functionality of the code after changing a few lines of code and checking for the output in the console in order to verify the hypothesis. Moreover, the researchers also computed a three-way transition probability among different AOIs to capture a longer sequence of behavior similar to the one captured by the two-way transition. For example, a three way transition "code–console–Variable View" could depict a behavior of a non-verified hypothesis about the functionality of the code and a subsequent attempt to experiment with different values of a variable under question.

Code reading patterns – Understanding the pre-written code is an essential part of debugging tasks. Sharma et. al [34] have shown in their experiment about program comprehension, that a specific way of reading the code is important for successful programming. When a programmer follows the data flow of a code written in a procedural/object oriented language, the eyes move mostly in vertical direction. However, when the code is being read as an English text, the eyes move mostly in horizontal manner. For the purpose of the experiment, the researchers computed the average "saccade horizontality" as a measure of the different reading patterns.

4.6 Data analysis

To address the research questions (RQs) presented in the Introduction section, the authors propose the following analysis to be conducted for each RQ:

(1) Considering the level of users' visual attention to the mirroring tool, a descriptive statistic will be used to compute the percentage of the total time spent looking at the EV.

(2) Regarding the relation between expertise, gaze, and debugging success, the authors will use Analysis of Variance (ANOVA) for

comparing the variables across different categories. For example, a one-way ANOVA will be conducted to test any potential differences between the performance levels of experts and novices. In addition, the authors will check the assumptions for ANOVA, and if they find variables which does not satisfy the homoscedasticity condition, a version of ANOVA will be used where homoscedasticity will not be assumed.

(3) To examine the relation between the gaze patterns on EV and the students' performance, the authors decided to measure a Pearson's correlation, which is about quantifying the strength of the relationship between variables, as both variables are continuous and not ranked.

(4) To find out what gaze patterns relate to debugging success, the authors decided to use linear models, with the debugging success as the dependent variable and the gaze patterns (time on AOIs, 2-way and 3-way transitions) as the process variables.

5 RESULTS

In this section the authors present the relations among the gaze patterns, the expertise, and the debugging success considering the study's research questions. [5]

Question 1. What is the level of students' visual attention to the mirroring tool (i.e. Exercise View)?

The results showed that the participants pay attention to the Exercise View (EV). The mean proportion of the overall time the participants spent on the EV is 14.8% (Std. Dev. = 5.3%). Figure 2 shows the distribution of the overall time spent on the EV by all of the participants.

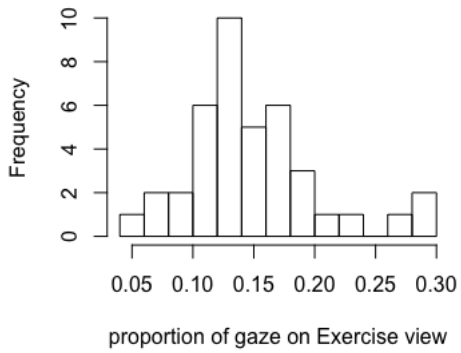

Figure 2: Proportion of gaze on the Exercise view by different students during debugging task.

Question 2. How students' expertise, success and gaze patterns are associated?

In this study, 25 of the participants were categorized as experts, while 15 participants were categorized as novices. The authors observed a significant difference between the performance levels of experts and novices. A one-way ANOVA without assuming the equal variances revealed that experts performed significantly better than novices ($F[1,36.77] = 15.09$, $p = 0.0004$, see Figure 3).

Tables 2, 3, and 4 show the relation between the debugging success and the gaze patterns. The results confirmed that these two

[5]We would like to point out here that in the rest of this paper whenever we mention "time spent", we consider the "proportion of overall time spent looking at".

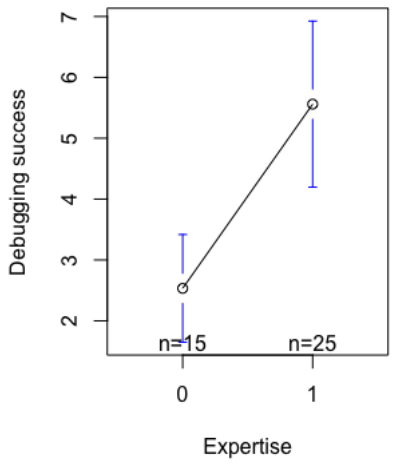

Figure 3: Debugging success for the different levels of expertise. The blue bar shows the 95% confidence intervals.

relations are similar to each other, hence in the rest of this section the authors present the analysis results from the relation describing the interaction of debugging success and the participants' gaze patterns.

Question 3. How does the time spent on the mirroring tool relate to the performance (i.e. code produced to solve a task)?

There is a significant negative correlation between the time spent on EV and the debugging success ($r(38) = -0.56$ $p = .0002$). The fact that the time spent on EV and the success are negatively correlated is not unexpected, since it is not important how many times participants looked at the EV, but how the information they perceived from the EV guided their further actions. Thus, that is why the authors aimed to observe two and three-way transitions among EV and the rest of the AOIs.

Question 4. What gaze patterns relate to high debugging success and what is the role of mirroring capabilities (i.e. Exercise view)?

In order to analyse the relation between the gaze and the performance, the authors considered the time spent on individual AOIs, the proportion of transitions among two AOIs (hereafter referred to as "2-way transitions") and the proportions of transitions among three AOIs (hereafter referred to as "3-way transitions"). Furthermore, the authors computed the average horizontality of the saccades while the students were reading the pre-existing code or what they had already written.

Table 2 shows the results from a linear model with the debugging success as the independent variable and the time spent on the individual AOIs as the process variables. Pearson's test verified that the time spent on EV is negatively correlated to the debugging success, while verified the relatively strong positive relation between the debugging success and the time spent on JUnit, console, problem, and variable view.

Since Pearson's correlation test demonstrated that the time spent on EV is negatively correlated with the success, the authors decided to analyze the gaze patterns as 2-way transitions to verify the hypothesis that, it is not the time spent on EV that contributes to the performance, but the steps taken after processing the information

Table 2: Proportion of time spent in each AOI and the relation with the debugging success

	Estimate	Error	t-value	Pr(>|t|)
Intercept	2.55	1.45	1.76	0.08
EV	-19.88	6.46	-3.07	**0.004**
JUnit	11.71	4.03	2.91	**0.006**
Console	9.25	4.35	2.13	**0.04**
Problem	13.32	4.60	2.89	**0.006**
Variable view	19.64	6.40	3.07	**0.004**

from the EV. Moreover, the authors also examined what gaze patterns mostly contributed to the debugging success. Table 3 shows the significant results from a linear model with the debugging success as the independent variable and the 2-way transitions as the process variables. The results demonstrated that the transitions involving EV (e.g. EV-JUnit, Console-EV, Problem-EV) explain a significant proportion of variance of the debugging success. To cross validate the results, the authors contrasted this model (Table 3) against a model without any EV related transitions and used ANOVA to compare the two models. The output from ANOVA showed a significant increase ($F[3, 28.54] = 6.80$, $p = 0.001$) in the AIC value of the model without EV transitions (149.75) as compared to the AIC value of the model with EV transitions (136.03). Consequently, these findings support the role of EV as a mirroring tool that could regulate basic behavior skills and improve performance.

Table 3: Transition probabilities (2-way transitions) and the relation with the debugging success

From AOI To AOI	Estimate	Error	t-value	P(>|t|)
Intercept	1.51	0.42	3.64	**0.0009**
EV to JU	21.05	7.62	2.76	**0.009**
Code to C	12.02	2.12	5.62	**0.00001**
C to EV	-11.58	5.64	-2.05	**0.05**
P to EV	-5.77	2.82	-2.05	**0.05**
JU to Code	6.50	2.02	3.22	**0.003**
VV to Code	16.27	4.75	3.43	**0.002**
C to Code	6.25	2.61	2.39	**0.02**

Next, the authors decided to examine the relation between 3-way transitions and the debugging success. The 3-way transitions capture longer sequence of user behavior that could point to possible new perspectives for interpretation of the relation. Table 4 shows the significant results from a linear model with the debugging success as the independent variable and the 3-way transitions as the process variables. The results demonstrated that the transitions involving EV (e.g. EV-JUnit-code, EV-Junit-variable) explain a significant proportion of variance of the debugging success. The authors contrasted this model (Table 3) against a model without any EV related transitions and used ANOVA to compare the two models. The output from ANOVA shows a significant increase ($F[2, 2.16] = 5.38$, $p = 0.01$) in the AIC value of the model without EV transitions (68.02) as compared to the AIC value of the model with EV transitions (59.75).

Table 4: Transition probabilities (3-way transitions) and the relation with the debugging success

From AOI1 To AOI2 To AOI3	Estimate	Error	t-value	P(>|t|)
Intercept	-0.91	0.19	-4.80	**0.00001**
EV to JU to Code	2.73	‘1.21	2.26	**0.03**
Code to C to VV	16.34	4.92	3.32	**0.002**
Code to VV to Code	1.55	0.74	2.08	**0.05**
VV to Code to VV	15.53	6.44	2.41	**0.02**
Code to C to Code	1.85	0.63	2.94	**0.006**
C to Code to C	1.74	0.52	3.37	**0.002**
EV to JU to VV	1.74	0.69	2.51	**0.02**
JU to Code to VV	4.77	1.92	2.48	**0.02**
JU to VV to Code	3.90	0.84	4.66	**0.00001**

Finally, the last part of the analysis aimed to examine the code reading patterns among the experts and the novices, and observe the level of understanding of the pre-written code. Hence, the authors computed the average horizontality of code reading saccades (specifically when students were not editing the code). The findings showed that experts had significantly more vertical saccades than the novices ($F[1, 30.8] = 6.85$, $p = 0.01$, see Figure 4) meaning that experts managed to demonstrated a better level of program comprehension. Moreover, saccade horizontality is also correlated with the debugging success ($r(38) = 0.67$, $p < 0.0001$, see Figure 5) confirming the fact that a specific way of reading the code is important for successful debugging.

6 DISCUSSION

The prime motivation of having a mirroring tool is to have a basic behavioral regulation as mirroring tools are known as awareness tools [18]. Mirroring tools offer the most basic level of support as the system simply reflects user's actions through graphical visualizations without processing the information. Increasing student's awareness of their own actions without abstracting or evaluating these actions, could help students to maintain representation of their progress and encourage them to enhance their metacognitive activities. Consequently, this study tried to orchestrate a behavioral regulation (e.g. visual attention, following instructions, working memory) of participants engaged in a debugging task. The behavioral regulation skills fall under the category of self-regulation,

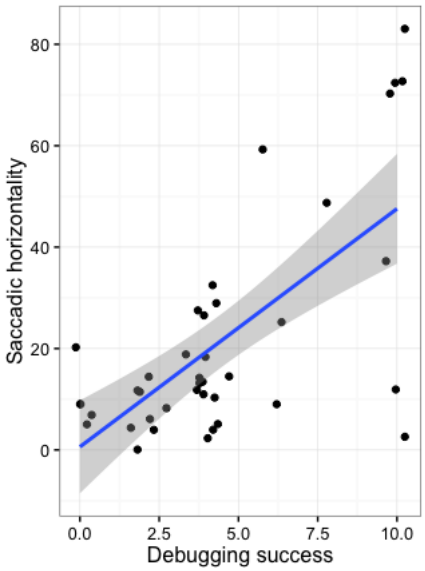

Figure 4: Debugging success and saccade horizontality. The blue line shows the linear model and the gray are shows the 95% confidence intervals.

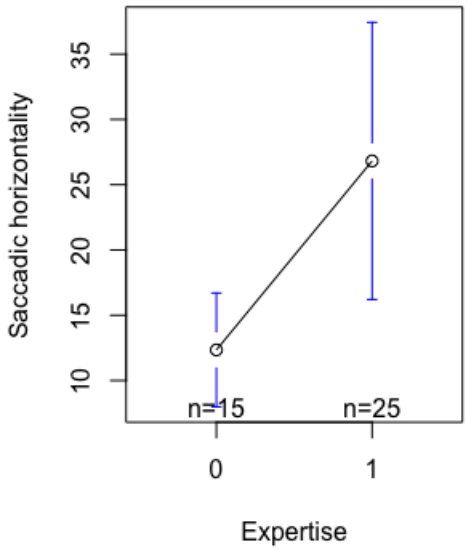

Figure 5: saccade horizontality for the different levels of expertise. The blue bar shows the 95% confidence intervals.

which is an essential feature for academic performance [39]. However, the question of how to enhance behavioral regulation skills within the academic context requires identification of mechanisms through which such interventions are most effective.

One such mechanism categorized as a mirroring tool (i.e. Exercise View) is suggested and presented in this study. The results from the study demonstrated that mirroring tools could regulate behavior depending on the contextual set up of the programming environment. With this in mind, the authors created a systematic debugging task that requires students to solve the task in a particular order to pass all of the unit tests and finish with success. The students who processed the information presented in the Exercise View (EV) and acted upon it, improved their performance and achieved higher level of debugging success than those who failed to process the information from EV. Moreover, experts were significantly more successful than novices. This result is consistent with the results reported from the studies mentioned in Section 2.4. However, what is more important regarding this distinction between novices and experts is the transitions they both performed among the different elements in IDE (e.g. AOIs), as these gaze patterns could be a potential input in developing relevant learning strategies based on expertise and knowledge level.

First of all, we considered the relation between the time spent on individual AOIs and the debugging success (hereafter referred to as "success"). The results show that time spent on EV was negatively correlated with success, while gaze on JUnit, console, problem, and variable view were positively correlated to success. This supports our claim that it is not the time that a student spends looking at EV important for success; but it is the time spent in processing the information from EV and acting upon it, which correlates to success. In order to figure out mistakes that a student made, (s)he had to check the unit test definition (JUnit) and the variable concerned (variable view). If the student did not look at this particular elements in IDE, his/her actions might suggest a hypothesis that the student is practicing trial and error attitude instead of a problem-solving strategy. Moreover, the gaze towards problem and console might suggest a hypothesis verification process opted by the student. To confirm this, we considered 2-way and 3-way transitions.

Considering 2-way transitions, we observed that EV to JUnit, code to console, console to code, variable view to code, and JUnit to code transitions were positively correlated with success. On the other hand, console to EV, and problem to EV transitions were negatively correlated with success. The positively correlated gaze transitions show a debugging behavior which corresponds to the following:

- understanding that there is a bug and finding the problem statement (EV to JUnit);
- locating the problem in the code (JUnit to code);
- trying to remove the bug while looking for corresponding output (code to console);
- going back to another part of the code to obtain the correct output (console to code);
- finding the variable causing the problem and locating it in the code (variable view to code).

On the other hand, the shifts from console to EV and problem to EV do not bring meaningful information to the debugging process. Hence, this could suggest that the student has a misconception regarding the role of EV by considering it as a feedback mechanism (e.g. trying to find answers in the EV) rather than a reflection mechanism (e.g. processing the information from EV and acting upon it). Another explanation might be that visual attention strategies among novices are not well developed as they are among experts.

Similarly, all significantly correlated 3-way transitions also demonstrated a positive correlation with success. Their corresponding debugging behavioral depictions are:

• understanding that there is a bug, finding the problem statement, and locating the problem in the code (EV to JUnit to code);
• understanding that there is a bug, finding the problem statement, and locating the variable causing the problem (EV to JUnit to variable view);
• trying to remove the bug while looking for corresponding output, and going back to another part of the code to obtain the correct output (code to console to code; console to code to console);
• finding the variable causing the problem, fixing it in the code, and repeating this for a few times before executing the code (code to variable view to code; variable view to code to variable view);
• finding the problem description, locating the variable causing the problem and/or locating the problem in the code (JUnit to code to variable view; JUnit to variable view to code).

These findings communicate the importance of EV as a mirroring tool in behavior regulation. Moreover, the authors decided to strengthen this finding by comparing two models to show how EV fits into the gaze patterns explaining success; one model with gaze patterns including EV, and another model without gaze patterns including EV. The results demonstrated that we loose a significant amount of information when we remove the gaze patterns including EV. This indicates the fact that information presented in EV and how students use this information is important for levels of success achieved.

On the other side, the significant negative correlation, between the saccadic horizontality and success, depicts that those who achieved higher success were also having more vertical saccades. This is highlighted by the finding of [34], from a pair program comprehension task.

All these findings fit in the aim of the study to observe a particular contextual set up of a programming environment in which researchers could detect what constitutes relevant data when performing debugging. Knowing what is essential in order to teach problem-solving skills, empowers teachers to create various learning strategies taking into account expertise and knowledge level. Moreover, having more AOIs than other studies (see Section 2.4) allows researchers to gain deeper understanding of user's cognitive needs and behavior patterns. Hence, greater understanding empowers easier regulation of behavioral skills and development of efficient tools and methodologies for teaching problem-solving skills.

7 CONCLUSION

The present eye-tracking study investigated the correlation between a mirroring tool developed in Eclipse and users' debugging success in a programming task. The aim of the study was to orchestrate a basic behavioral regulation of participants engaged in a debugging task. Thus, 40 computer science majors were given 40 minutes to solve 5 debugging tasks presented as a part of the main method of the main class of a 100 lines of Java code. The results have demonstrated that the gaze patterns of successful debuggers corresponded with attention shifts among EV and other AOIs (i.e. console, problem, and JUnit). The fact that the time users spent on EV and their success was negatively correlated, proves that it is not important how long and how many times participants looked at the EV, but how the information they perceived from the EV guided their further actions. This fact was further examined with 2-way and 3-way transitions among EV and the rest of the AOIs. The results from the analysis confirmed that users who processed the information from EV and displayed debugging behavior as presented in Table 3 and Table 4, correlates to successful completion of the debugging task.

Due to the encouraging results from this study, the authors plan to continue the research with the mirroring tool by including a more controlled study to investigate the availability of a mirroring tool and its relation with the debugging success, as well as the motivational aspects of the tool. Another interesting aspect to consider in future studies is the notion of mirroring tools as instruments that trigger deeper form of interactive learning in terms of cognitive effort, where the users can interact with the information presented in real-time. Finally, as a general acknowledgment comes the fact that human decision making is a part of successful analytics solution as any other technical component [40]. Therefore, it is very important to engage participants in designing and developing a pedagogically sound and user-centred future learning environments. To achieve this goal, researchers and designers need to follow and practice user-centred design approach while utilizing user-centred analytics to support their actions.

8 ACKNOWLEDGEMENTS

This work was supported by the Research Council of Norway under the project FUTURE LEARNING (255129/H20).