JOURNAL OF LEARNING ANALYTICS

SOLAR SOCIETY for LEARNING ANALYTICS RESEARCH

Volume 5(3), 98–119. http://dx.doi.org/10.18608/jla.2018.53.7

Gaze-Driven Design Insights to Amplify Debugging Skills: A Learner-Centred Analysis Approach

Katerina Mangaroska[1], Kshitij Sharma[2], Michail Giannakos[3], Hallvard Træteberga[4], Pierre Dillenbourg[5]

Abstract

This study investigates how multimodal user-generated data can be used to reinforce learner reflection, improve teaching practices, and close the learning analytics loop. In particular, the aim of the study is to utilize user gaze and action-based data to examine the role of a mirroring tool (i.e., Exercise View in Eclipse) in orchestrating basic behavioural regulation during debugging. The results demonstrated that students who processed the information presented in the Exercise View and acted upon it, improved their performance and achieved a higher level of success than those who failed to do so. The findings shed light on what constitutes relevant data within a particular learning context in programming using gaze patterns. Moreover, these findings could guide the collection of essential learner-centred analytics for designing usable, modular learning environments based on data-driven approaches.

Notes for Practice

- Mirroring tools could regulate learner behaviour depending on the contextual set up of the programming environment.
- This study demonstrated that students who improved their performance and achieved a higher level of debugging success have gaze patterns that corresponded with attention shifts with the mirroring tool and other areas of interest (AOIs).
- Visual attention strategies among novices are not as well developed as they are among experts. This is shown in the successful and unsuccessful debugging behaviour patterns calculated from two- and three-way transition probabilities that we observed.
- IDE-based (integrated development environment) learning analytics combined with gaze data could aid researchers and educators in designing and delivering interventions to enhance student progress by guiding them to attend to the right information at the right time so as to maximize student understanding of relevant concepts.

Keywords

Eye-tracking, learner-centred analytics, mirroring tools, behaviour regulation, debugging, multimodal analytics

Submitted: 10.06.2018 — **Accepted:** 31.07.2018 — **Published:** 11.12.2018

Corresponding author [1] Corresponding author. Email: mangaroska@ntnu.no, Address: Department of Computer Science, Norwegian University of Science and Technology, Sem Sælands vei 9, IT-bygget, 7034, Trondheim, Norway, ORCID ID 0000-0002-7853-0429

[2] Email: kshitij.sharma@ntnu.no, Address: Department of Computer Science, Norwegian University of Science and Technology, Sem Sælands vei 9, IT-bygget, 7034, Trondheim, Norway

[3] Email: michailg@ntnu.no, Address: Department of Computer Science, Norwegian University of Science and Technology, Sem Sælands vei 9, IT-bygget, 7034, Trondheim, Norway, ORCID ID 0000-0002-8016-6208

[4]Email: hal@ntnu.no, Address: Department of Computer Science, Norwegian University of Science and Technology, Sem Sælands vei 9, IT-bygget, 7034, Trondheim, Norway

[5] Email: pierre.dillenbourg@epfl.ch, Address: School of Computer and Communication Sciences, École Polytechnique Fédérale de Lausanne, Rolex Learning Center (RLC) D1 740, 1015, Lausanne, Switzerland

1. Introduction

To gain greater understanding of user needs and behaviours, researchers should utilize the available multifaceted user-generated data, which will empower them to design technology that amplifies human learning. More specifically, they should employ various modalities of user-centred analytics to improve learning, predict behaviour, and advise users and educators by

converting educational data into meaningful information. However, most educational systems still have concerns about what constitutes relevant data (Verbert et al., 2014), and fail to consider user perspectives or actively involve users in the design and development of learning experiences. Moreover, the majority of past studies (Mangaroska & Giannakos, 2017) focus on data extraction based on convenience and availability (e.g., click-stream data from learning management systems), rather than on the richness and objectivity of data that sometimes cannot be captured verbally, with logging actions, or by simple observation. Thus, extensive research in psychophysiology, linking psychological and cognitive states and processes to various measures, already exists (Kramer, 1990; Rowe, Sibert, & Irwin, 1998), but is less focused on programming. However, more insights have been generated into perceptions of task difficulty using psychophysiology measures (Fritz, Begel, Müller, Yigit-Elliott, & Züger, 2014), working memory and language processing in program comprehension (Siegmund et al., 2014), finding defects in source code (Sharif, Falcone, & Maletic, 2012), and comprehending different representations of code (Sharif & Maletic, 2010; Bednarik, 2012).

Hence, this study aims to use eye-tracking to gain insights into user debugging behaviour (e.g., how users 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 user. The idea behind the mirroring tool is to raise awareness of the user's own actions and reinforce reflective learning. Thus, this paper contributes to the knowledge base of learning-centred analysis by providing evidence about how mirroring tools can orchestrate behaviour regulation skills (e.g., visual attention, working memory) within the academic context. Although gaze data has already proven its potential and value in understanding how students learn to program and debug (Sharma, Jermann, Nüssli, & Dillenbourg, 2012), 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 in the user's debugging behaviour and to empower multimodal user-centred analysis to design relevant learning strategies for programming/debugging. As a result, this study addresses the following research questions:

RQ1: What is the level of student visual attention to the mirroring tool (i.e., Exercise View)?
RQ2: How are student expertise, success, and gaze patterns related?
RQ3: How does time spent on the mirroring tool relate to performance (i.e., code produced to solve a task)?
RQ4: What gaze patterns relate to high debugging success and what is the role of mirroring capabilities (i.e., Exercise view)?

The rest of paper is structured as follows: Section 2 outlines the related research; Section 3 presents our research objectives; Section 4 presents our methodology; Section 5 presents the empirical results. Section 6 discusses the results derived, while Section 7 summarizes the contribution of this paper.

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 amounts of user-generated data are 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 and design user-centred learning environments that are usable and underpin a robust learning experience, there is an ongoing challenge to converge techniques and methods from interdisciplinary fields such as software engineering, cognitive sciences, and technology enhanced learning (TEL; Balacheff & Lund, 2013). To tackle this goal, researchers need to utilize learning analytics to inform learning designs, and develop data-driven tools and learning strategies that will enhance teaching practices and student learning experiences (Lockyer & Dawson, 2011; Mangaroska & Giannakos, 2017; Siemens, 2012).

Most of the current educational systems focus on the educators' view and rarely involve students in the design and development processes. For instance, a literature study (de Quincey, Turner, Williams, & Kyriacou, 2016) documented that from 22 learning analytics related systems, only five are designed to be used entirely by students, and only few are available for general use. Martinez-Maldonado et al. (2015) employed a five-stage LATUX workflow 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 (2014) proposed a pedagogical intervention design concept that emphasizes the importance of enhancing existing learning analytics practices in the design of user-centred teaching and learning activities considering new technologies. Dawkins (2016) 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 user needs. Thus, selecting and deciding how to employ different learning analytics to support user-

centred design is a challenging task that 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 observing what constitutes relevant data when learners engage in problem-solving tasks (i.e., debugging), as this information could provide valuable input in designing user-centred learning experiences for teaching and learning programming using gaze data.

2.2. IDE-Based Learning Analytics

In computing education, students spend most of their time working on programming assignments in integrated development environments (IDEs). Thus, it is logical for educators and researchers to use IDEs to collect continuous streams of data in order to make sense of learner actions and behaviours, and thereby improve teaching and learning (Altadmri & Brown, 2015). A legacy of research depicts early attempts to study learners' programming progress by using think-aloud protocols (Ericsson & Simon, 1980; Kessler & Anderson, 1986), code snapshots (Bonar & Soloway, 1983), keystroke-level data (Guzdial, 1994), and log data (Goldenson & Wang, 1991; Jadud, 2006). Nowadays, the functionality of many IDEs can be extended with plug-ins to collect data with additional features, lifting the previous technological restrictions. Examples include student–IDE interactions (Brown, Kolling, McCall, & Utting, 2014; Hundhausen, Olivares, & Carter, 2017), asynchronous discussion posts (Carter & Hundhausen, 2015), automatic testing against test cases (Edwards & Perez-Quinones, 2008), survey/quiz data to gain insights into learner attitudes and conceptual understanding of tasks (Ihantola, Sorva, & Vihavainen, 2014), studies collecting data from eye-trackers (Busjahn et al., 2014; Kevic et al., 2015; Mangaroska, Sharma, Giannakos, Trætteberg, & Dillenbourg, 2018), mouse and keyboard pressure (Arapakis, Lalmas, & Valkansas, 2014; Begel, 2016), heart rate (Ahonen et al., 2016), and electro-dermal activity (Müller, 2015). Although data with additional features can help explain how students learn to program, to our knowledge, none of the hardware devices used to collect data from learners engaged in programming tasks is directly integrated with the IDE, which again imposes limitations on the range of data that can be collected. However, researchers have collected and analyzed various analytics from IDEs, ranging from very low level (e.g., click-stream interactions, including mouse movements and keystrokes) to higher level (e.g., line-edits to source code and IDE actions). Furthermore, different statistical predictors were used to predict learner performance, such as Error Quotient (Jadud, 2006) and Watwin Score (Watson, Li, & Godwin, 2013), focusing on compilation errors; Normalized Programming State Model (NPSM) focusing on editing, compilation, and debugging behaviours (Carter, Hundhausen, & Adesope, 2015); and learner programming behaviours using a series of programming states (Blikstein et al., 2014).

Considering the state-of-the-art with respect to IDE plug-ins and data collection tools, Hundhausen et al. (2017) presented a 4-step cyclical process model for IDE-based learning analytics — 1) collect data, 2) analyze data, 3) design intervention, and 4) deliver intervention — that can be followed to explore the additional sources of data that have not yet been investigated. Thus, for the data collection for this study, the authors considered the taxonomy of standard programming data that can be automatically collected from Eclipse IDE, and the taxonomy of data augmented with additional features and functionality (i.e., gaze data). Regarding the data analysis, this study used two of the six different techniques proposed by Hundhausen et al. (2017; i.e., counting raw data points and visualization of data from IDE-based analytics). Furthermore, in this study Eclipse IDE was also used to raise awareness of the user's own actions and reinforce reflective learning by utilizing the Eclipse plug-in (i.e., Exercise View) that visualizes learner actions while they work on programming tasks. The authors consider this as a step towards the last two stages in the cyclical process model for IDE-based learning analytics, design, and delivery of intervention to enhance student progress.

2.3. Tools Usage within Computer-Based Learning Environments

Past studies show that students are not passive recipients of instructional design decisions because they control their choices, considering their preferences and preconceptions (Winne, 1982, 2006; Lust, Elen, & Clarebout, 2013a). Doyle (1977) and Winne (1982) introduced the cognitive mediation paradigm and demonstrated that students respond differently to instructional design decisions (e.g., tools). Furthermore, Lust et al. (2013a) showed that students are responsible for shaping their learning path while regulating tools usage throughout the course. In general, computer-based learning environments introduced different tools that support and shape student learning. For example, information tools outline the learning resources and compensate for students' lack of domain knowledge; scaffolding tools guide students to reach their learning goals and compensate for their lack of cognitive and metacognitive strategies; mirroring tools reflect user actions to aid them in maintaining representation of their progress and encourage them to enhance their metacognitive activities; and cognitive tools induce higher order thinking skills (Lust, Elen, & Clarebout, 2013b). Tools are important for developing knowledge and skills, regulating behaviour (e.g., visual attention, following instructions, working memory), and enriching and self-regulating learning practices (Hoskins & van Hoof, 2005). However, tool usage is influenced by students' task perceptions, their skill in using the tool, and their motivation (Perkins, 1985). Consequently, this study tries to orchestrate the behavioural regulation (e.g., visual attention, following instructions, working memory) of participants engaged in a debugging task using a mirroring tool. In addition, visual attention strategies have not been sufficiently explored regarding their contribution to learning

programming. Therefore, this study utilizes a mirroring tool to explore this issue in a lab experiment with students who already have domain knowledge and basic-to-intermediate cognitive and metacognitive skills in programming.

2.4. Eye-Tracking and Learning Technologies

In computer-based learning environments, the text is still the core component (Rayner, 2009). However, visualizations such as drawings, images, diagrams, and videos are usually associated with the text, imposing attentional demands on learners that are not always helpful during the learning process (e.g., increasing the cognitive load without any learning gain). For written textual information in computer-based learning environments, a hierarchical structure is especially helpful for learners with low prior knowledge (Amadieu, Van Gog, Paas, Tricot, & Mariné, 2009), as is the case with novices in programming for whom attention can be guided to the main concepts on which they need to focus. Moreover, if the textual information is complemented by visualizations such as images or diagrams, learners are exposed to the split-attention effect or spatial contiguity effect (Ayres & Sweller, 2005). In these learning scenarios, eye-tracking data can aid researchers and educators in studying attention and shifts in attention underlying the split-attention effect. Past research has shown that under experimental conditions, learners focus on text; under naturalistic conditions, they focus on images, and when the information is appropriately integrated, then learners process it all (Jarodzka, Janssen, Kirschner, & Erkens, 2015). Moreover, eye-tracking data has shown that as learners increase their knowledge and expertise they tend to fixate faster on more relevant information (Charness, Reingold, Pomplun, & Stampe, 2001; Jarodzka, Scheiter, Gerjets, & Van Gog, 2010). This can aid researchers and educators in creating models of learning that will guide learners to attend to the right information at the right time in order to maximize their comprehension of relevant concepts. The aforementioned research complements the cognitive theory of multimedia learning, which builds on the philosophy of designing e-learning environments focusing on cognitive theories of how people learn and on scientifically valid research in multimedia learning and human–computer interaction (Mayer, 2005). Eye-tracking can be used to enhance this philosophy, by utilizing information coming from our neurological processes, unveiling the way we process information, and allowing us to improve the learning environment design to amplify our learning and capacities.

2.5. Eye-Tracking and User-Centred Design

Eye-tracking helps researchers gain insights into user behaviour (e.g., how users process information or interact with visual information) that cannot be captured verbally, via other more ordinal user-data (e.g., click-streams), or by observation (Cooke, 2005). Moreover, in educational technology, developers and researchers must design intuitive systems with high user experience and learnability (i.e., learning experience; Garreta Domingo & Mor Pera, 2007). In order to accomplish this, the systems need to follow user-centred design guidelines, maintain minimal errors, and avoid user-frustration (Bojko, 2013).

In eye-tracking studies, saccades and fixations are the main indicators of user behaviour for interacting and processing information. Previous studies (Ali-Hasan, Harrington, & Richman, 2008; Garrard, 2014; Granka, Joachims, & Gay, 2004; Tzafilkou & Protogeros, 2017) used eye-tracking data to evaluate and improve the design and the user experience in different development stages. In one study, Garrard (2014) reported that a high number of saccades indicates a long time to search for information, while a high number of fixations represents a high degree of user uncertainty. In a different study (Ramakrisnan, Jaafar, Razak, & Ramba, 2012), 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 Mele and Federici's (2012) idea that eye-tracking is a "psychotechnology" because it emphasizes the intra-systemic perspective of the relationship between the user and the technology. These studies show that eye-tracking provides unique access to user behaviour (in educational contexts, learners), hence one can use learner gaze data to extract information and feed the data back to the learners while efficiently closing the learning analytics (LA) loop (Clow, 2012). The key step in closing the LA loop is to feed back the generated data by or about learners (output) to the same group of learners through interventions that alter the learning process (input). Thus, as eye-tracking is considered an objective technique that can provide insights into many aspects of human cognitive abilities, such as problem solving, reasoning, and mental strategies (Ball, Lucas, Miles, & Gale, 2003; Just & Carpenter 1976; Yoon & Narayanan, 2004), gaze data can be utilized to inform the design of learning technologies by effectively covering a wide range of user needs and behaviours.

As can be seen from past studies (Duchowski, 2002; Mele & Federici, 2012), eye-tracking is a promising methodology for collecting data, measuring user behaviour, and calculating the amount of processing (e.g., ease of use, perceived playfulness, cognitive load) when users interact with computer systems. This relevant way of observing the dynamic trace of where user attention is directed in relation to the visual information presented means that designers can embrace and practice user-centred design.

2.6. Eye-Tracking and Problem Solving

Most results show that eye-tracking allows researchers to gather attentional data for users while they perform tasks. Likewise, previous studies (Harbluk, Noy, Trbovich, & Eizenman, 2007; Jones, 2003; Kaller, Rahm, Bolkenius, & Unterrainer, 2009; Reingold, Charness, Pomplun, & Stampe, 2001) had shown that eye-tracking can explain various constructs like contextual expertise, task dependency, and task complexity.

Reingold et al. (2001) 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 a larger area of visual span and were faster at detecting modifications in the original game configurations than in the random ones, while there were no differences found between novices and intermediate players. Moreover, the researchers reported that experts do encode larger chunks of the configuration than the rest of the players due to their use of the foveal and parafoveal regions of the eye (Chase & Simon, 1973).

Considering the relationship between task dependency and gaze patterns, Kaller et al. (2009) studied the time course of visuospatial problem solving. During the experiment, 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 on 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 (i.e., time between the presentation of the problem and the onset of the first action), most subjects directed their fixations mainly to the left side irrespective of the state arrangement. As a result, the authors discovered a strong dependency between personal preference and gaze pattern. 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 of internalization, planning, execution, and verification. Another relevant study was carried out to compare car drivers driving while performing (or not performing) arithmetic tasks (Harbluk et al., 2007). The results showed that drivers pay less attention to the mirrors, the 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 regarding cognitive load, reduction of safety, and distraction increased from “no task” to “easy task” to “difficult task” conditions. Jones (2003) used a car park problem to find the relationship between problem solving processes and gaze data. The goal of the car park problem is to maneuver a car out of a parking space. The parking lot has other cars as well, which can be moved only in their initial orientation. The authors looked at the fixation time three moves prior to the car move and three moves after the car move. The fixation time on the problem was longer for the car move than for the prior or succeeding moves. Moreover, non-solvers spent significantly more time on the free area than the solvers.

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

2.7. Eye-Tracking and Debugging

Previous eye-tracking studies (Bednarik, 2012; Bednarik & Tukiainen, 2004a, 2004b; Bednarik & Tukiainen, 2008) show a clear relationship between gaze patterns and task-based performance in debugging. Bednarik and Tukiainen (2004b) conducted a study of restricted focus viewer (RFV) against a control condition (without RFV) in a debugging task. An RFV is a special tool that blurs parts of the screen to make participants focus on a specific area of interest (AOI). The results showed no significant difference between the two experimental conditions, but displayed a relationship between debugging success and expertise. In terms of gaze behaviour, the RFV condition induced more switches from the code area to the output area than the control condition (Bednarik & Tukiainen, 2004a). In a different study, Bednarik and Tukiainen (2004b) investigated the relationship between expertise, gaze for code visualization, and the debugging success of programmers. There was a relationship between expertise and success, but the results were non-conclusive regarding the use of visualization in the debugging task. Finally, in a recent study, Bednarik (2012) 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, Sharif et al. (2012) compared the first scan time (i.e., the time taken by the participants to read the code for the first time) 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 behaviour, this study showed that successful debuggers had a more vertical gaze than those who performed 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 debug. However, the current understanding of the role of visual attention in program comprehension or debugging and coordination of representations is still at an early stage. Previous studies (Bednarik, 2012; Bednarik & Tukiainen, 2004b; Bednarik & Tukiainen, 2008; Stein & Brennan, 2004) have used 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 to 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 a 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 fails to capture the last phase of the debugging process. Therefore, instead of just finding the bugs (as it was the case in Bednarik, 2012; Bednarik & Tukiainen, 2004b; Bednarik & Tukiainen, 2008; Sharif et al., 2012), participants in the present study were also required to remove those bugs.

2. **We have a more systematic way of debugging in the 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 period. This was not the case with the experiments reported in Section 2.4. Moreover, in most of the studies, participants were provided with a sample code and one instance of the output (Bednarik, 2012; Bednarik & Tukiainen, 2004b; Romero, Cox, du Boulay, & Lutz, 2002; Romero, Cox, du Boulay, & Lutz, & Bryant, 2007), and they could not edit the code. As we pointed out in the previous paragraph, not letting the participants debug the program hinders their understanding of the debugging process.

3. We designed **a mirroring tool** (a plug-in to Eclipse) to reflect progress to the user, as well as success or failure during the debugging task. Success or failure was based on the number of unit tests passed by each participant.

4. Previous studies (Bednarik & Tukiainen, 2004b; Romero et al., 2002) used an external graphical representation of the code (data flow, control flow, class-object diagrams) to aid in participant understanding. However, this representation might hinder the understanding of the code if the participant had no previous knowledge of using representations in learning programming, as it was evident from the reported gaze patterns; participants did not pay much attention to the visual representation. In this study, **we used the "variable view"** provided by Eclipse IDE to show the state of the different selected variables during debugging. Participants could choose when to enable this view and when to remove it from the screen if they found it redundant.

5. In terms of **eye-tracking analysis**, the saliency comes from the previous four points. Since we allowed participants to edit the code and execute it as many times as they wanted, there were frequent switches in the code, errors, and output areas of the IDE. Moreover, frequent switches were observed between the code and the unit test panels. When participants enabled the "variable view" to get 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, and output) and a few used a fourth one, called sequence. In our study, we decided to define seven main AOIs (i.e., code, exercise view, Junit test, problem, console, variable view, and debug view) to gain a better understanding of user cognitive skills. Additionally, for completion we added two more AOIs based on the IDE: toolbar and project explorer. Table 1 presents a detailed description of the AOIs. Finally, during the analysis stage, we calculated two- and three-way transitions for greater understanding of user cognitive needs, as well as for higher validity and reliability of the study; previous studies reported in Section 2.4 analyzed only two-way transitions.

Table 1. Descriptions of the AOIs in Eclipse IDE

Area of Interest (AOI)	Description
Toolbar	The toolbar of the IDE
Variable View	During a debug, allows changing the value of a variable to test how your program handles a particular value or to speed through a loop
DebugView	Allows managing the debugging or running of a program in the workbench
ProjectExplorer	Provides a hierarchical view of the artefacts in the workbench
Junit	Allows listing the unit tests to be passed by the main Java class
Code	Code panel where the code is written
ExerciseView	Allows seeing the coding, saving, testing, and progress
Problem	Shows the errors and/or warnings raised by the Java Compiler
Console	Shows the output of the code

Consequently, the research objective of this paper is threefold. First, the authors wanted to gain more insight into how users process information in debugging tasks and how they interact with visual information utilizing a mirroring tool. For this purpose, the authors designed a debugging task in Eclipse IDE that was also extended with a plug-in (i.e., the mirroring tool Exercise View) that collects data with additional features while students program/debug. Second, the authors wanted to examine the role of Exercise View in debugging, and collect rich and objective multimodal data that can be used practically and effectively to better prepare future curricula and design learning strategies considering expertise and knowledge proficiency. Finally, the authors wanted to gain more understanding in behaviour regulation for developing efficient tools and methodologies for teaching problem-solving skills.

4. Methodology

4.1. Debugging Activity

The authors designed a debugging activity in conjunction with their partners from École Polytechnique Fédérale de Lausanne. The main task assigned to participants was debugging a Java class named Person, which 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 his list of children, as shown in Figure 1. Participants could check the correctness of the code by running the provided unit tests.

4.2. Participants

During the spring of 2017, an experiment was performed at a contrived computer lab setting at 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 their integrated development environment (IDE). Moreover, they were also familiar with the built-in debugging tool provided by Eclipse. The focus of this study is to examine how user-generated gaze data can be used to reinforce student reflective practices. Moreover, the study also considered whether students can practice problem-solving strategies (e.g., debugging a code) coupled with reflection (from a mirroring tool) rather than using trial and error.

4.3. Procedure

Upon arrival at the laboratory, 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 for each participant's 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, participants were asked to perform a pre-task, which required removing 90 errors from a skeleton code within 10 minutes. After this task, participants were given 40 minutes to solve five debugging tasks presented as part of the main method of the main class of 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, the authors recorded and later analyzed participants' fixations (i.e., the state when the eye remains still over a time) and saccades (i.e., the rapid motion of the eye from one fixation to another; Holmqvist et al., 2011). For their participation in the experiment, participants were rewarded with CHF 25 (25 Swiss Francs).

Figure 1. Consistencies absent from the original version of the code provided to participants in the present study. There were a few consistency checks implemented as unit tests. 1) Gender consistency: the mother should be female and the father should be male. 2) Child–parent consistency: if Jens is the child of Merit, Merit should be the mother of Jens, and vice-versa. 3) The removal of a child–parent relationship from either a parent or a child should also apply to the whole family. 4) Adoption consistency: the child–parent (addition and removal) and the gender consistencies should be maintained in the case of adoption.

4.4. The Mirroring Tool

For the purpose of the experiment, students used the Eclipse IDE to complete the exercise. The exercise had pre-written unit tests and five questions that led 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 2). The data that this plug-in collected and mirrored back to the students included the following: 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 work incrementally 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 the EV reflects. This information could later be used to implement a user-centred approach in designing learning strategies for programming and debugging.

ISSN 1929-7750 (online).

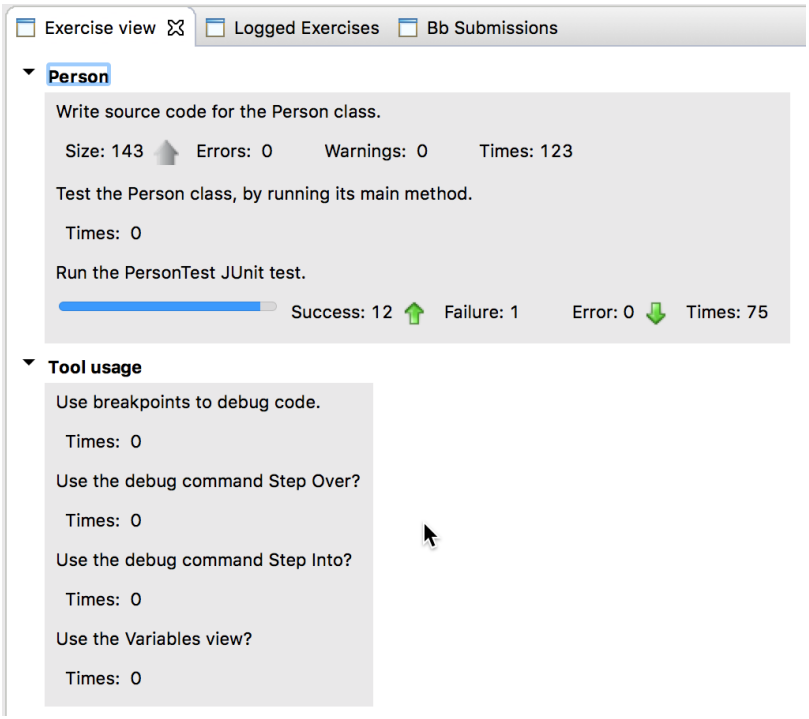

Figure 2. The Exercise View configured for the experiment. The top panel shows the progress of the "Person" class. The first line shows the lines of code, the number of errors and warnings by the Java compiler, and the number of times the code has been compiled. The second line shows the number of times 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 colour shows whether the change is considered 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 a pre-task test, where students were presented with a task of removing 90 errors from a skeleton code. The simple instruction was to write a 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 five 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" (N = 25) or "novices" (N = 10) based on the minimalism in their code (the fewest lines of code written and compiled without error).

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 a few bugs in otherwise complete code that would make the code fail all ten unit tests. In order to pass all of the unit tests, the students were required to solve the debugging exercise 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 was taken as the measure of "debugging success."

Individual areas of interest (AOIs): Eclipse IDE was divided into nine AOIs (see Table 1). During the analysis, the researchers computed the proportion of time spent on each AOI as well as transitions between the different AOIs. The results for the AOIs were later compared to participant performance and expertise.

Transitions between 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 the IDE to another, which might correspond to a specific behavioural pattern while students were debugging the code. For example, the transition from Code to Console might depict the behaviour of having a hypothesis about the functionality of the code after changing a few lines and checking the output in the console in order to verify the hypothesis. Moreover, the researchers also computed a three-way transition probability between different AOIs to capture a longer sequence of behaviour similar to the one captured by the two-way transition. For example, a three-way

ISSN 1929-7750 (online).

transition "Code–Console–Variable View" could depict a behaviour of a non-verified hypothesis about the functionality of the code and a subsequent attempt to experiment with different values of the variable in question.

Code reading patterns: Understanding the pre-written code is an essential part of a debugging task. Sharma et al. (2012) 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 a vertical direction. However, when the code is being read as an English text, the eyes move mostly in a horizontal manner. For the purpose of the experiment, the researchers computed the average "saccade horizontality" as a measure of different reading patterns.

4.6. Data Analysis

To address the research questions (RQs) presented in Section 1, the authors propose the following analysis to be conducted for each RQ:

RQ1: Considering the level of user visual attention to the mirroring tool, a descriptive statistic will be used to compute the percentage of total time spent looking at the EV.

RQ2: Regarding the relationship between expertise, gaze, and debugging success, the authors will use analysis of variance (ANOVA) for comparing 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 that do not satisfy the homoscedasticity condition, a version of ANOVA will be used where homoscedasticity will not be assumed. This version of ANOVA uses the Welch (1951) correction for F statistics (effect size).

RQ3: To examine the relationship between gaze patterns on the EV and student performance, the authors decided to measure a Pearson's correlation, which quantifies the strength of the relationship between variables, as both variables are continuous and not ranked.

RQ4: To find out which gaze patterns relate to debugging success, the authors decided to use linear models, with debugging success as the dependent variable and gaze pattern (time on AOIs, 2-way and 3-way transitions) as the process variable.

5. RESULTS

In this section, the authors present the relationships between gaze pattern, expertise, and debugging success, considering the study's research questions.

First, to gain understanding of the selected variables, a descriptive statistic was performed; the results are presented in Table 2. Next, the data was checked for normality. A Shapiro-Wilk test was performed due to the sample size (n=40). The results showed that the data has normal distribution: p values were not significant, the values for skewness are within the range of –1 to 1, and the values for kurtosis are in the range of –2 to 2. Next, to investigate the relationship between the variables, the authors performed a pair-wise correlation analysis between the individual AOIs and the two- and three-way transitions. A Pearson's rank correlation was used to compute the correlations of the variables. The results from the correlation analysis are presented in Tables 3, 4, and 5.

RQ1. What is the level of student visual attention to the mirroring tool (i.e., Exercise View)?

The results showed that participants pay attention to the Exercise View (EV). The mean proportion of the overall time that participants spent on the EV is 14.8% (Std. Dev. = 5.3%). Figure 3 shows the distribution of overall time spent on the EV by all participants.

RQ2. How are student expertise, success, and gaze patterns related?

In this study, 25 participants were categorized as experts, while 15 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 4). Tables 6, 7, and 8 show the relationship between debugging success and gaze pattern. Table 9 shows the relationship between expertise and gaze pattern. Experts exhibit gaze patterns that correlate to high debugging performance more often than novices. For example, the two-way transition probability of "Junit to Code" is positively correlated with debugging success and experts exhibit this pattern significantly more often than novices. Other results confirmed that these two relationships (gaze and success; gaze and expertise) are similar, hence, in the rest of this section, the authors describe the interaction between participants' debugging success and gaze pattern.

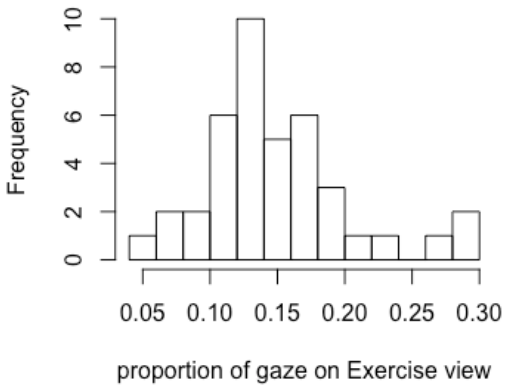

Figure 3. Proportion of gaze on the EV by different students during the debugging task.

Table 2. Proportion of Time Spent in Each AOI and Relationship with Debugging Success

	Variable	Mean	SD
Individual AOI	EV	0.14	0.05
	Junit	0.14	0.07
	Console	0.12	0.07
	Problem	0.11	0.06
	Variable	0.02	0.05
2-way transitions	EV.Junit	0.01	0.02
	code.console	0.11	0.10
	console.EV	0.01	0.03
	problem.EV	0.03	0.07
	Junit.code	0.11	0.12
	variable.code	0.02	0.04
	console.code	0.08	0.10
3-way transitions	ev.junit.code	0.12	0.08
	ev.junit.variable	0.15	0.13
	code.console.variable	0.07	0.05
	code.variable.code	0.06	0.10
	code.console.code	0.31	0.22
	variable.code.variable	0.01	0.01
	console.code.console	0.22	0.15
	junit.code.variable	0.14	0.11
	unit.variable.code	0.39	0.24

Table 3. Correlation Matrix between Individual AOI

AOI		a1	a2	a3	a4	a5
EV	**a1**	–	–	0.12	**–**	**–**
			0.04		**0.42****	**0.33***
Junit	**a2**	–	–	0.09	0.02	–0.07
Console	**a3**	–	–	–	–0.06	**0.44****
Problem	**a4**	–	–	–	–	0.20
Variable	**a5**	–	–	–	–	–

* $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$

Table 4. Correlation Matrix between Two-Way Transitions

Transitions		2t1	2t2	2t3	2t4	2t5	2t6	2t7
EV.Junit	**2t1**	–	–0.05	–0.16	–0.20	0.09	0.04	–0.02
code.console	**2t2**	–	–	–0.01	–0.07	**0.35***	0.31	**0.46****
console.EV	**2t3**	–	–	–	0.002	–0.23	–0.21	–0.27
problem.EV	**2t4**	–	–	–	–	**–0.40***	–0.04	**–0.33***
Junit.code	**2t5**	–	–	–	–	–	**0.41****	**0.57*****
variable.code	**2t6**	–	–	–	–	–	–	0.23
console.code	**2t7**	–	–	–	–	–	–	–

* p < 0.05; ** p < 0.01; *** p < 0.001

Table 5. Correlation Matrix between Three-Way Transitions

Transitions		3t1	3t2	3t3	3t4	3t5	3t6	3t7	3t8	3t9
ev.junit.code	3t1	–	0.29	0.62***	0.15	0.45**	0.39**	0.17	0.52***	0.52***
ev.junit.variable	3t2	–	–	0.58***	0.14	0.50***	0.14	–0.06	0.57***	0.58***
code.console.variable	3t3	–	–	–	0.12	0.08	0.04	0.22	0.09	0.09
code.variable.code	3t4	–	–	–	–	0.17	–0.29	0.09	0.12	0.12
code.console.code	3t5	–	–	–	–	–	0.03	0.19	0.08	0.07
variable.code.variable	3t6	–	–	–	–	–	–	0.05	0.25	0.22
console.code.console	3t7	–	–	–	–	–	–	–	0.10	0.40**
junit.code.variable	3t8	–	–	–	–	–	–	–	–	0.09
unit.variable.code	3t9	–	–	–	–	–	–	–	–	–

* p < 0.05; ** p < 0.01; *** p < 0.001

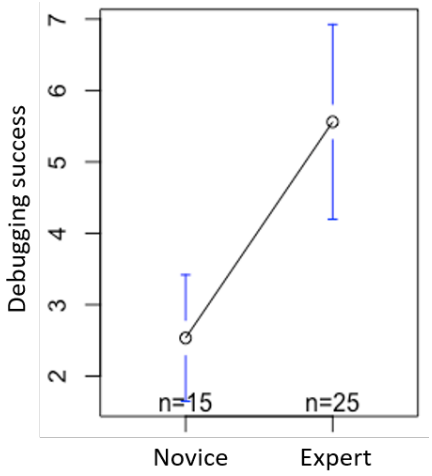

Figure 4. Debugging success for different levels of expertise. The blue bar shows the 95% confidence intervals.

Table 6. Testing the Effect of Time Spent in Each AOI with Debugging Success

AOI	Estimate	Error	t-value	p (>|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	3007	0.004

Table 7. Transition Probabilities (Two-Way Transitions) and Relationship to Debugging Success

From AOI to AOI	Estimate	Error	t-value	p (>|t|)
Intercept	1.51	0.42	3.64	0.0009
EV to Junit	21.05	7.62	2.76	0.009
Code to Console	12.02	2.12	5.62	0.00001
Console to EV	–11.58	5.64	–2.05	0.05
Problem to EV	–5.77	2.82	–2.05	0.05
Junit to Code	6.50	2.02	3.22	0.003
Variable View to Code	16.27	4.75	3.43	0.002
Console to Code	6.25	2.61	2.39	0.02

Table 8. Transition Probabilities (Three-Way Transitions) and Relationship with Debugging Success

From AOI to AOI to AOI	Estimate	Error	t-value	p (>|t|)
Intercept	–0.91	0.19	–4.80	0.0000
Exercise View to Junit to Code	2.73	1.21	2.26	0.03
Code to Console to Variable View	16.34	4.92	3.32	0.002
Code to Variable View to Code	1.55	0.74	2.08	0.05
Variable View to Code to Variable View	15.53	6.44	2.41	0.02
Code to Console to Code	1.85	0.63	2.94	0.006
Console to Code to Console	1.74	0.52	3.37	0.002
Exercise View to Junit to Variable View	1.74	0.69	2.51	0.02
Junit to Code to Variable View	4.77	1.92	2.48	0.02
Junit to Variable View to Code	3.90	0.84	4.66	0.00001

Table 9. Relationship between Expertise and Different Gaze Patterns

Gaze pattern	df1	df2	F	p
Exercise View	1	32.43	3.04	0.09
Junit	1	37.95	< 1	NS*
Console	1	33.67	1.78	0.19
Problem	1	23.14	2.45	0.13
Exercise View to Junit	1	21.91	< 1	NS
Code to Console	1	30.41	4.01	0.05
Console to Exercise View	1	31.24	< 1	NS
Problem to Exercise View	1	20.28	3.55	0.07
Junit to Code	1	36.83	6.08	0.01
Variable View to Code	1	26.18	6.45	0.01
Console to Code	1	37.32	7.21	0.01
Exercise View to Junit to Code	1	29.12	2.42	0.13
Code to Console to Variable View	1	36.43	13.66	0.0007
Code to Variable View to Code	1	29.76	< 1	NS
Variable View to Code to Variable View	1	32.97	6.29	0.01
Code to Console to Code	1	37.19	9.94	0.003
Console to Code to Console	1	27.16	1.15	0.29
Exercise View to Junit to Variable View	1	37.79	6.42	0.01
Junit to Code to Variable View	1	32.67	15.94	0.0003
Junit to Variable View to Code	1	37.137	14.31	0.0005

Note: These results are obtained from one-way ANOVA, without assuming equal variance between the two groups. *NS = Not Significant

RQ3. How does time spent on the mirroring tool relate to performance (i.e., code produced to solve a task)?

There is a significant negative correlation between time spent on the EV and debugging success ($r(38) = -0.56$ $p = .0002$). The fact that time spent on the EV and 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. This is why the authors aimed to observe two- and three-way transitions between the EV and the rest of the AOIs.

RQ4. What gaze patterns relate to high debugging success and what is the role of mirroring capabilities (i.e., Exercise view)?

In order to analyze the relationship between gaze and performance, the authors considered the time spent on individual AOIs, the proportion of transitions between two AOIs (hereafter referred to as "two-way transitions"), and the proportions of transitions between three AOIs (hereafter referred to as "three-way transitions").

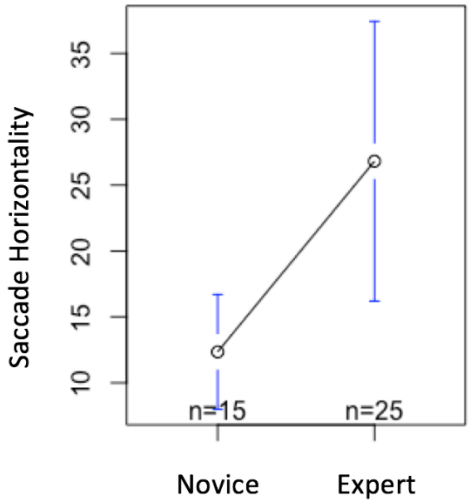

Figure 5. Saccade horizontality for expertise. The blue bar shows the 95% confidence intervals.

Furthermore, the authors computed the average horizontality of the saccades while the students were reading the pre-existing code or what they had written. Table 6 shows the results from a linear model with debugging success as the independent variable and time spent on individual AOIs as the process variables. Pearson's test verified that the time spent on the EV is negatively correlated with debugging success, while it also verified the relatively strong positive relationship between debugging success and time spent on Junit, Console, Problem, and Variable View.

Since Pearson's correlation test demonstrated that time spent on the EV is negatively correlated with success, the authors decided to analyze gaze patterns as two-way transitions to verify the hypothesis that it is not the time spent on the EV that contributes to performance, but the steps taken after processing the information from the EV. Moreover, the authors also examined which gaze patterns mostly contributed to debugging success.

Table 7 shows the significant results from a linear model with debugging success as the independent variable and two-way transitions as the process variables. The results demonstrated that transitions involving the EV (e.g., EV–Junit, Console–EV, and Problem–EV) explain a significant proportion of variance of debugging success. To cross validate the results, the authors contrasted this model (Table 7) against a model without any EV related transitions and used ANOVA to compare the two models. The output from the 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 the EV as a mirroring tool that could regulate basic behaviour skills and improve performance.

Next, the authors decided to examine the relationship between three-way transitions and debugging success. The three-way transitions capture longer sequences of user behaviour that could point to possible new perspectives for interpreting the relationship. Table 8 shows the significant results from a linear model with debugging success as the independent variable and three-way transitions as the process variables. The results demonstrated that transitions involving the EV (e.g., EV–Junit–Code, EV–Junit–Variable View) explain a significant proportion of variance of debugging success. The authors contrasted this model (Table 8) against a model without any EV related transitions and used ANOVA to compare the two models. The output from the 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).

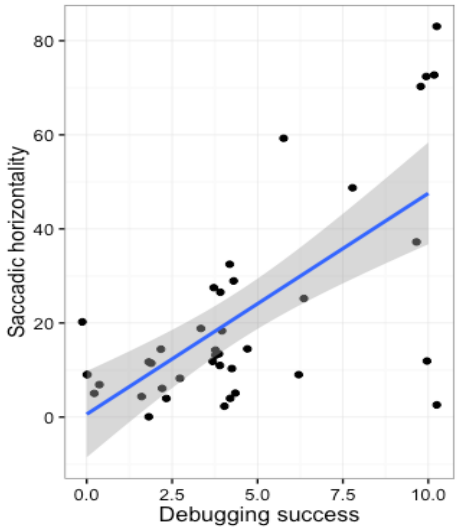

Figure 6. Debugging success and saccade horizontality. The blue line shows the linear model and the grey area shows the 95% confidence intervals.

Finally, the last part of the analysis aimed to examine the code reading patterns between experts and novices, and observe the level of understanding of 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 novices ($F [1, 30.8] = 6.85$, $p = 0.01$; see Figure 5) meaning that experts managed to demonstrate a better level of program comprehension. Moreover, saccade horizontality is also correlated with debugging success ($r (38) = 0.67$, $p < 0.0001$; see Figure 6) confirming the fact that a specific way of reading code is important for successful debugging.

6. Discussion

The prime motivation for having a mirroring tool is to have basic behavioural regulation, as mirroring tools are known as awareness tools (Gutwin & Greenberg, 1999). Mirroring tools offer the most basic level of support as the system simply reflects user actions through graphical visualizations without processing the information. Increasing student 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 the behavioural regulation (e.g., visual attention, following instructions, working memory) of participants engaged in a debugging task. Behavioural regulation skills fall under the category of self-regulation, which is an essential feature of academic performance (Vohs & Baumeister, 2016). However, the question of how to enhance behavioural 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 behaviour depending on the contextual set up of the programming environment. The authors tried to reduce the extraneous cognitive load when designing the debugging activity by following three design principles, as suggest by Hundhausen et al. (2017). 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 a higher level of debugging success than those who failed to process the information from the 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 the IDE (e.g., AOIs), as these gaze patterns could be a potential input in developing relevant learning strategies based on expertise and knowledge level. In addition, researchers and educators could combine the continuous streams of data they collect from the IDE with gaze patterns, and utilize the insights into informing learning designs of programming environments based on knowledge and skills. Moreover, they could develop data-driven tools to enhance teaching and learning practices, as well as design and delivery of interventions to aid student progress.

In this study, the authors started by investigating the correlation between the variables and found out that the EV has a moderate but significant positive relationship with the Variable View and Problem AOI. Regarding the two-way transitions,

there is a weak positive relationship between looking at the errors and/or warnings raised by the Java Compiler and the ones shown in the EV, and locating the problem in the code. While for three-way transitions, there is a moderate positive relationship between understanding that there is a bug, finding the problem statement, and locating the bug in the code with trying to remove the bug while looking for corresponding output and finding the variable causing the problem. Next, we considered the relationship between time spent on individual AOIs and debugging success (hereafter referred to as "success"). The results show that time spent on the EV was negatively correlated with success, while the gaze on JUnit, Console, Problem, and Variable View were positively correlated to success. This supports our claim that the time spent processing information from the EV and acting upon it is more important for success than simply the time that a student spends looking at the EV. In order to figure out what mistakes a student made, they had to check the unit test definition (Junit) and the variable concerned (Variable View). If the student did not look at this particular element in the IDE, their actions might suggest that the student is practicing trial and error instead of a problem-solving strategy. Moreover, the gaze towards Problem and Console might suggest a hypothesis verification process by the student. To confirm this, we considered two-way and three-way transitions to explore the visual attention strategies in the debugging process and how they differ across levels of expertise.

Considering two-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. Thus, the positively correlated gaze transitions show a debugging behaviour that 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)

This sequence of actions captured using eye-tracking from students' visual attention shifts is the desired debugging behaviour that educators should teach students. If educators know this sequence of actions, they can design learning tasks that will regulate and support desired student behaviour in learning and practicing problem-solving skills. 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 the 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 the EV and acting upon it). Another explanation might be that visual attention strategies among novices are not as well developed as they are among experts.

Similarly, all significantly correlated three-way transitions also demonstrated a positive correlation to success. Their corresponding debugging behavioural depictions are as follows:

- 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)

In addition, the authors also calculated the distinction between novices and experts in the transitions they both performed among the different AOIs (i.e., elements in the IDE), as this distinction should be considered in developing programming/debugging strategies based on knowledge proficiency. Thus, when comparing the behaviour patterns of experts versus novices, two-way gaze transitions show a debugging behaviour that corresponds to the following:

- 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)

Likewise, when comparing the behaviour patterns of experts versus novices, the three-way gaze transitions show a debugging behaviour that corresponds to the following gaze transitions:

- Understanding that there is a bug, finding the problem statement, and locating the variable causing the problem (EV to Junit to Variable View)
- Locating the variable causing the problem, fixing the problem, looking at the change in the variable's value at each step of iteration (Code to Console 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)
- Finding the variable causing the problem, fixing it in the code, and repeating this for a few times before executing the 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)

As can be observed from the results, EV-related transitions are not significantly different between experts and novices; they are only related to success. Moreover, all these patterns are exhibited by experts significantly more than novices. In addition, one interesting finding that needs to be underlined is that novices do not have transitions to/from variable view. This supports the assumption that visual attention strategies among novices are not as well developed as they are among experts, or novices do not feel confident to change the value of a variable to test how the program handles a particular value, so they practice trial and error instead.

These findings communicate the importance of the EV as a mirroring tool in behaviour regulation and of eye-tracking as a promising technology for measuring user behaviour in computer-based environments. Moreover, the authors decided to strengthen this finding by comparing two models to show how the EV fits into the gaze patterns explaining success: one model with gaze patterns including the EV, and another model without gaze patterns including the EV. The results demonstrated that we lose a significant amount of information when we remove the gaze patterns including the EV. This indicates the fact that information presented in the EV, and how students use this information, is important for the level of success achieved. This is only one data-driven example that shows how multimodal user-centred analysis can empower researchers and educators to design learning models for programming/debugging to guide students about which relevant information to attend to when trying to increase student comprehension of important concepts and improve their learning progress. Moreover, the results from this dynamic observation of students' visual attention strategies using eye-tracking is a practice that educators and designers can embrace to practice user-centred design of programming assignments that could stimulate active, deeper, self-regulated learning.

On the other side, the significant negative correlation between saccade horizontality and success shows that those who achieved higher success also had more vertical saccades. This is highlighted by the findings of Sharma et al. (2012), from a pair-program comprehension task. This finding supports the work of Amadieu et al. (2009) that hierarchical structure in computer-based learning environments is especially helpful for novices.

All these findings fit with 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 users' cognitive needs and behaviour patterns. Hence, greater understanding empowers easier regulation of behavioural skills and development of efficient tools and methodologies for teaching problem-solving skills.

7. Conclusions

The present eye-tracking study investigated the relationship between a mirroring tool developed in Eclipse and users' debugging success in a programming task. Thus, 40 computer science majors were given 40 minutes to solve five debugging tasks presented as part of the main method of the main class of 100 lines of Java code. The results demonstrate that the gaze patterns of successful debuggers corresponded with attention shifts between the EV and other AOIs (i.e., Console, Problem, and Junit). The fact that the time users spent on the EV was negatively correlated with their success 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 two-way and three-way transitions between the EV and the rest of the AOIs. The results from the analysis confirmed that users who processed the information from the EV and displayed debugging behaviour, as presented in Tables 3 and 4, correlate 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 relationship with 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 forms of interactive learning in terms of cognitive effort, where users can interact with the information presented in real-time. Finally, in general, human decision making is as much a part of successful analytics solutions as any other technical component (Wise, 2014). Therefore, it is very important to engage participants in designing and developing pedagogically sound, user-centred future learning environments. To achieve this goal, researchers and designers must follow and practice user-centred design approaches while utilizing user-centred analytics to support their actions.

Declaration of Conflicting Interest

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

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

References

Ahonen, L., Cowley, B., Torniainen, J., Ukkonen, A., Vihavainen, A., & Puolamäki, K. (2016). Cognitive collaboration found in cardiac physiology: Study in classroom environment. *PLoS ONE, 11*(7). http://dx.doi.org/10.1371/journal.pone.0159178

Ali-Hasan, N. F., Harrington, E. J., & Richman, J. B. (2008). Best practices for eye tracking of television and video user experiences. *Proceedings of the 1st International Conference on Designing Interactive Uer Experiences for TV and Video* (uxTV '08), 22–24 October 2008, Silicon Valley, CA, USA (pp. 5–8). New York: ACM. http://dx.doi.org/10.1145/1453805.1453808

Altadmri, A., & Brown, N. C. C. (2015). 37 million compilations: Investigating novice programming mistakes in large-scale student data. *Proceedings of the 46th ACM Technical Symposium on Computer Science Education* (SIGCSE '15), 4–7 March 2015, Kansas City, MO, USA (pp. 522–527). New York: ACM. http://dx.doi.org/10.1145/2676723.2677258

Amadieu, F., Van Gog, T., Paas, F., Tricot, A., & Mariné, C. (2009). Effects of prior knowledge and concept- map structure on disorientation, cognitive load, and learning. *Learning and Instruction, 19*, 376–386. http://dx.doi.org/10.1016/j.learninstruc.2009.02.005

Arapakis, I., Lalmas, M., & Valkansas, G. (2014). Understanding within-content engagement through pattern analysis of mouse gestures. *Proceedings of the 23rd ACM International Conference on Information and Knowledge Management* (CIKM '14), 3–7 November 2014, Shanghai, China (pp. 1439–1448). New York: ACM. http://dx.doi.org/10.1145/2661829.2661909

Ayres, P., & Sweller, J. (2005). The split-attention principle in multimedia learning. In R. E. Mayer (Ed.), *The Cambridge handbook of multimedia learning* (pp. 135–146). New York: Cambridge University Press.

Balacheff, N., & Lund, K. (2013). Multidisciplinarity vs. multivocality, the case of learning analytics. *Proceedings of the 3rd International Conference on Learning Analytics and Knowledge* (LAK '13), 8–12 April 2013, Leuven, Belgium (pp. 5–13). New York: ACM. http://dx.doi.org/10.1145/2460296.2460299

Ball, L. J., Lucas, E. J., Miles, J. N., & Gale, A. G. (2003). Inspection times and the selection task: What do eye-movements reveal about relevance effects? *The Quarterly Journal of Experimental Psychology*, *56*(6), 1053–1077. http://dx.doi.org/10.1080/02724980244000729

Bednarik, R. (2012). Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human–Computer Studies*, *70*(2), 143–155. http://dx.doi.org/10.1016/j.ijhcs.2011.09.003

Bednarik, R., & Tukiainen, M. (2004a). Visual attention and representation switching in java program debugging: A study using eye movement tracking. *Proceedings of the 16th Annual Workshop of the Psychology of Programming Interest Group* (PPIG 2004), 5–7 April 2004, Carlow, Ireland (pp. 159–169).

Bednarik, R., & Tukiainen, M. (2004b). Visual attention tracking during program debugging. *Proceedings of the 3rd Nordic*

Conference on Human–Computer Interaction (NordiCHI04), 23–27 October 2004, Tampere, Finland (pp. 331–334). New York: ACM. http://dx.doi.org/10.1145/1028014.1028066

Bednarik R., & Tukiainen, M. (2008). Temporal eye-tracking data: Evolution of debugging strategies with multiple representations. *Proceedings of the 2008 Symposium on Eye Tracking Research and Applications* (ETRA '08), 26–28 March 2008, Savannah, GA, USA (pp. 99–102). New York: ACM. http://dx.doi.org/10.1145/1344471.1344497

Begel, A. (2016). Fun with software developers and biometrics: Invited talk. *Proceedings of the 1st International Workshop on Emotion Awareness in Software Engineering* held at the 38th International Conference on Software Engineering (ICSE '16), 14–22 May 2016, Austin, TX, USA (pp. 1–2). New York: ACM. http://dx.doi.org/10.1145/2897000.2897007

Blikstein, P., Worsley, M., Piech, C., Gibbons, A., Sahami, M., & Cooper, S. (2014). Programming pluralism: Using learning analytics to detect patterns in novices' learning of computer programming. *International Journal of the Learning Sciences*, *23*(4), 561–599. http://dx.doi.org/10.1080/10508406.2014.954750

Bojko, A. (2013). Eye tracking the user experience: A practical guide to research. Brooklyn, NY: Rosenfeld Media.

Bonar, J., & Soloway, E. (1983). Uncovering principles of novice programming. *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages* (POPL '83), 24–26 January 1983, Austin, Texas, USA (pp. 10–13). New York: ACM. http://dx.doi.org/10.1145/567067.567069

Brown, N. C. C., Kolling, M., McCall, D., & Utting, I. (2014). Blackbox: A large scale repository of novice programmers' activity. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (SIGCSE '14), 5–8 March 2014, Atlanta, Georgia, USA (pp. 223–228). New York: ACM. http://dx.doi.org/10.1145/2538862.2538924

Busjahn, T., Schulte, C., Sharif, B., Begel, A., Hansen, M., Bednarik, R., Orlov, P., Ihantola, P., Shchekotova, G., & Antropova, M. (2014). Eye tracking in computing education. *Proceedings of the 10th Annual Conference on International Computing Education Research* (ICER '14), 11–13 August 2014, Glasgow, United Kingdom (pp. 3–10). New York: ACM. http://dx.doi.org/10.1145/2632320.2632344

Carter, A. S., & Hundhausen, C. D. (2015). The design of a programming environment to support greater social awareness and participation in early computing courses. *Journal of Computing Sciences in Colleges, 31*(1), 143–153.

Carter, A. S., Hundhausen, C. D., & Adesope, O. (2015). The normalized programming state model: Predicting student performance in computing courses based on programming behaviour. *Proceedings of the 11th Annual Conference on International Computing Education Research* (ICER '15), 9–13 July 2015, Omaha, Nebraska, USA (pp. 141–150). New York: ACM. http://dx.doi.org/10.1145/2787622.2787710

Charness, N., Reingold, E. M., Pomplun, M., & Stampe, D. M. (2001). The perceptual aspect of skilled performance in chess: Evidence from eye movements. *Memory & Cognition, 29*, 1146–1152.

Chase, W. G., & Simon, H. A. (1973). Perception in chess. *Cognitive Psychology*, *4*(1), 55–81. http://dx.doi.org/10.3758/BF03206384

Clow, D. (2012). The learning analytics cycle: Closing the loop effectively. In S. Buckingham Shum, D. Gašević, & R. Ferguson (Eds.), *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge* (LAK '12), 29 April–2 May 2012, Vancouver, BC, Canada (pp. 134–138). New York: ACM. http://dx.doi.org/10.1145/2330601.2330636

Cooke, L. (2005). Eye tracking: How it works and how it relates to usability. *Technical Communication*, *52*(4), 456–463.

Dawkins, R. (2016). Content strategy: A lesson from the industry for university learning analytics. In S. Barker, S. Dawson, A. Pardo, & C. Colvin (Eds.), *Show Me the Learning: Proceedings of the 33rd Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education* (ASCILITE 2016). 28–30 November 2016, Wellington, New Zealand (pp. 172–181). Australasian Society for Computers in Learning in Tertiary Education.

de Quincey, E., Turner, M., Williams, N., & Kyriacou, T. (2016). Learner analytics: The need for user-centred design in learning analytics. *EAI Endorsed Transactions on Ambient Systems*, *16*(9). doi:http://dx.doi.org/10.4108/eai.23-8-2016.151643

Doyle, W. (1977). Paradigms for research on teacher effectiveness. *Review of Research in Education*, *5*, 392–431. http://dx.doi.org/10.3102/0091732X005001163

Duchowski, A. T. (2002). A breadth-first survey of eye-tracking applications. *Behaviour Research Methods, Instruments, & Computers*, *34*(4), 455–470. http://dx.doi.org/10.3758/BF03195475

Edwards, S. H., & Perez-Quinones, M. A. (2008). Web-CAT: Automatically grading programming assignments. *Proceedings of the 13th Annual Conference on Innovation and Technology in Computer Science Education* (ITiCSE '08), 30 June–2 July 2008, Madrid, Spain (pp. 328–328). New York: ACM. http://dx.doi.org/10.1145/1384271.1384371

Ericsson, K. A., & Simon, H. A. (1980). Verbal reports as data. *Psychological Review*, *87*(3), 215.

http://dx.doi.org/10.1037/0033-295X.87.3.215

Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., & Züger, M. (2014). Using psycho-physiological measures to assess task difficulty in software development. *Proceedings of the 36th International Conference on Software Engineering* (ICSE '14), 31 May–7 June 2014, Hyderabad, India (pp. 402–413). New York: ACM. http://dx.doi.org/10.1145/2568225.2568266

Garrard, W. (2014). An examination of eyetracking artifacts related to website design for individuals with cognitive disability. *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics* (SMC2014), 5–8 October 2014, San Diego, CA, USA (pp. 3197–3202). IEEE Computer Society. http://dx.doi.org/10.1109/SMC.2014.6974420

Garreta Domingo, M., & Mor Pera, E. (2007). User centered design in e-learning environments: From usability to learner experience. http://hdl.handle.net/10363/600

Goldenson, D. R., & Wang, B. J. (1991). Use of structure editing tools by novice programmers. *Empirical Studies of Programmers: Fourth Workshop* (pp. 99–120). Ablex.

Granka, L. A., Joachims, T., & Gay, G. (2004). Eye-tracking analysis of user behaviour in www search. *Proceedings of the 27th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval* (SIGIR '04), 25–29 July 2004, Sheffield, United Kingdom (pp. 478–479). New York: ACM. http://dx.doi.org/10.1145/1008992.1009079

Gutwin, C., & Greenberg, S. (1999). The effects of workspace awareness support on the usability of real-time distributed groupware. *ACM Transactions on Computer–Human Interaction, 6*(3), 243–281. http://dx.doi.org/10.1145/329693.329696

Guzdial, M. (1994). Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments, 4*(1), 1–44. http://dx.doi.org/10.1080/1049482940040101

Harbluk, J. L., Noy, Y. I., Trbovich, P. L., & Eizenman, M. (2007). An on-road assessment of cognitive distraction: Impacts on drivers' visual behaviour and braking performance. *Accident Analysis & Prevention*, *39*(2), 372–379. http://dx.doi.org/10.1016/j.aap.2006.08.013

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H., & Van de Weijer, J. (2011). *Eye tracking: A comprehensive guide to methods and measures*. Oxford, UK: Oxford University Press.

Hoskins, S. L., & van Hooff, J. C. (2005). Motivation and ability: Which students use online learning and what influence does it have on their achievement? *British Journal of Educational Technology*, *36*(2), 177–192. http://dx.doi.org/10.1111/j.1467-8535.2005.00451.x

Hundhausen, C. D., Olivares, D. M., & Carter, A. S. (2017). IDE-based learning analytics for computing education: A process model, critical review, and research agenda. *ACM Transactions on Computing Education*, *17*(3), 11. http://dx.doi.org/10.1145/3105759

Ihantola, P., Sorva, J., & Vihavainen, A. (2014). Automatically detectable indicators of programming assignment difficulty. *Proceedings of the 15th Annual Conference on Information Technology Education* (SIGITE/RIIT'14), 15–18 October 2014, Atlanta, GA, USA (pp. 33–38). New York: ACM. http://dx.doi.org/10.1145/2656450.2656476

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. *Proceedings of the 2nd International Workshop on Computing Education Research* (ICER '06), 9–10 September 2006, Canterbury, United Kingdom (pp. 73–84). New York: ACM. http://dx.doi.org/10.1145/1151588.1151600

Jarodzka, H., Janssen, N., Kirschner, P. A., & Erkens, G. (2015). Avoiding split attention in computer-based testing: Is neglecting additional information facilitative? *British Journal of Educational Technology*, *46*(4), 803–817. http://dx.doi.org/10.1111/bjet.12174

Jarodzka, H., Scheiter, K., Gerjets, P., & Van Gog, T. (2010). In the eyes of the beholder: How experts and novices interpret dynamic stimuli. *Learning and Instruction*, *20*, 146–154.

Jones, G. (2003). Testing two cognitive theories of insight. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, *29*(5), 1017. http://dx.doi.org/10.1016/j.learninstruc.2009.02.019

Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive Psychology*, *8*(4), 441–480. http://dx.doi.org/10.1016/0010-0285(76)90015-3

Kaller, C. P., Rahm, B., Bolkenius, K., & Unterrainer, J. M. (2009). Eye movements and visuospatial problem solving: Identifying separable phases of complex cognition. *Psychophysiology*, *46*(4), 818–830. http://dx.doi.org/10.1111/j.1469-8986.2009.00821.x

Kessler, C. M., & Anderson, J. R. (1986). A model of novice debugging in LISP. In E. Soloway & S. Iyengar (Eds.), *Papers presented at the 1st workshop on Empirical Studies of Programmers,* 5–6 June 1986, Washington, DC, USA (pp. 198–212). Norwood, NJ: Ablex Publishing.

Kevic, K., Walters, B. M., Shaffer, T. R., Sharif, B., Shepherd, D. C., & Fritz, T. (2015). Tracing software developers' eyes and interactions for change tasks. *Proceedings of the 10th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering* (ESEC/FSE'15), 31 August–4 September 2015, Bergamo, Italy (pp. 202–213). New York: ACM. http://dx.doi.org/10.1145/2786805.2786864

Kramer, A. F. (1990). Physiological metrics of mental workload: A review of recent progress. Technical Report NPRDC-TN-90-23, Navy Personnel Research and Development Center, San Diego, CA.

Lockyer, L., & Dawson, S. (2011). Learning designs and learning analytics. In P. Long, G. Siemens, G. Conole, & D. Gašević (Eds.), *Proceedings of the 1st International Conference on Learning Analytics and Knowledge* (LAK '11), 27 February–1 March 2011, Banff, AB, Canada (pp. 153–156). New York: ACM. http://dx.doi.org/10.1145/2090116.2090140

Lust, G., Elen, J., & Clarebout, G. (2013a). Regulation of tool-use within a blended course: Student differences and performance effects. *Computers & Education*, *60*(1), 385–395. http://dx.doi.org/10.1016/j.compedu.2012.09.001

Lust, G., Elen, J., & Clarebout, G. (2013b). Students' tool-use within a web enhanced course: Explanatory mechanisms of students' tool-use pattern. *Computers in Human Behaviour*, *29*(5), 2013–2021. http://dx.doi.org/10.1016/j.chb.2013.03.014

Mangaroska, K., & Giannakos, M. (2017). Learning analytics for learning design: Towards evidence-driven decisions to enhance learning. *Proceedings of the 12th European Conference on Technology Enhanced Learning* (EC-TEL 2017), 12–15 September 2017, Tallinn, Estonia (pp. 428–433). Lecture Notes in Computer Science, Springer. http://dx.doi.org/10.1007/978-3-319-66610-5_38

Mangaroska, K., Sharma, K., Giannakos, M., Trætteberg, H., & Dillenbourg, P. (2018). Gaze insights into debugging behaviour using learner-centred analysis. *Proceedings of the 8th International Conference on Learning Analytics and Knowledge* (LAK '18), 5–9 March 2018, Sydney, NSW, Australia (pp. 350–359). New York: ACM. http://dx.doi.org/10.1145/3170358.3170386

Martinez-Maldonado, R., Pardo, A., Mirriahi, N., Yacef, K., Kay, J., & Clayphan, A. (2015). The LATUX workflow: Designing and deploying awareness tools in technology enabled learning settings. *Proceedings of the 5th International Conference on Learning Analytics and Knowledge* (LAK '15), 16–20 March 2015, Poughkeepsie, NY, USA (pp. 1–10). New York: ACM. http://dx.doi.org/10.1145/2723576.2723583

Mayer, R. E. (Ed.). (2005). *The Cambridge handbook of multimedia learning*. New York: Cambridge University Press.

Mele, M. L., & Federici, S. (2012). A psycho technological review on eye-tracking systems: Towards user experience. *Disability and Rehabilitation: Assistive Technology*, *7*(4), 261–281. http://dx.doi.org/10.3109/17483107.2011.635326

Müller, S. C. (2015). Measuring software developers' perceived difficulty with biometric sensors. *Proceedings of the 37th International Conference on Software Engineering* (ICSE '15), 16–24 May 2015, Florence/Firenze, Italy (pp. 887–890). Piscataway, NJ: IEEE Press. http://dx.doi.org/10.1109/ICSE.2015.284

Perkins, D. (1985). The fingertip effect: How information-processing technology shapes thinking. *Educational Researcher*, *14*(7), 11–17. http://dx.doi.org/10.3102/0013189X014007011

Ramakrisnan, P., Jaafar, A., Razak, F. H. A., & Ramba, D. A. (2012). Evaluation of user interface design for leaning management system (LMS): Investigating student's eye tracking pattern and experiences. *Procedia: Social and Behavioural Sciences*, *67*, 527–537. http://dx.doi.org/10.1016/j.sbspro.2012.11.357

Rayner, K. (2009). Eye movements and attention in reading, scene perception, and visual search. *The Quarterly Journal of Experimental Psychology*, *62*, 1457–1506. http://dx.doi.org/10.1080/17470210902816461

Reingold, E. M., Charness, N., Pomplun, M., & Stampe, D. M. (2001). Visual span in expert chess players: Evidence from eye movements. *Psychological Science*, *12*(1), 48–55. http://dx.doi.org/10.1111/1467-9280.00309

Romero, P., Cox, R., du Boulay, B., & Lutz, R. (2002). Visual attention and representation switching during java program debugging: A study using the restricted focus viewer. In M. Hegarty, B. Meyer, & N. H. Narayanan (Eds.), *Diagrammatic Representation and Inference: Diagrams 2002. Lecture Notes in Computer Science, 2317.* Berlin/Heidelberg: Springer. http://dx.doi.org/10.1007/3-540-46037-3_23

Romero, P., Cox, R., du Boulay, B., Lutz, R., & Bryant, S. (2007). Debugging strategies and tactics in a multi-representation software environment. *International Journal of Human–Computer Studies*, *65*(12), 992–1009. http://dx.doi.org/10.1016/j.ijhcs.2007.07.005

Rowe, D. W., Sibert, J., & Irwin, D. (1998). Heart rate variability: Indicator of user state as an aid to human–computer interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '98), 4–9 April 2009, Boston, MA, USA (pp. 480–487). Washington, DC: Human Factors in Computing Systems. New York: ACM Press/Addison-Wesley. http://dx.doi.org/10.1145/274644.274709

Sharif, B., Falcone, M., & Maletic, J. (2012). An eye-tracking study on the role of scan time in finding source code defects. *Proceedings of the 2012 Symposium on Eye Tracking Research and Applications* (ETRA '12), 28–30 March 2012, Santa Barbara, CA, USA (pp. 381–384). New York: ACM. http://dx.doi.org/10.1145/2168556.2168642

Sharif, B., & Maletic, J. (2010). An eye tracking study on camelcase and under_score identifier styles. *Proceedings of the 18th International Conference on Program Comprehension* (ICPC 2010), 30 June–2 July 2010, Braga, Minho, Portugal. IEEE Computer Society. http://dx.doi.org/10.1109/ICPC.2010.41

Sharma, K., Jermann, P., Nüssli, M. A., & Dillenbourg, P. (2012). Gaze evidence for different activities in program understanding. *Proceedings of the 24th Annual Conference of the Psychology of Programming Interest Group* (PPIG 2012) 21–23 November 2012, London, UK. http://ppig.org/library/paper/gaze-evidence-different-activities-program-understanding

Siegmund, J., Kästner, C., Apel, S., Parnin, C., Bethmann, A., Leich, T., Saake, G., & Brechmann, A. (2014). Understanding source code with functional magnetic resonance imaging. *Proceedings of the 36th International Conference on Software Engineering* (ICSE '14), 31 May–7 June 2014, Hyderabad, India (pp. 378–389). New York: ACM. http://dx.doi.org/10.1145/2568225.2568252

Siemens, G. (2012). Learning analytics: envisioning a research discipline and a domain of practice. In S. Buckingham Shum, D. Gašević, & R. Ferguson (Eds.), *Proceedings of the 2nd International Conference on Learning Analytics and Knowledge* (LAK '12), 29 April–2 May 2012, Vancouver, BC, Canada (pp. 4–8). New York: ACM. http://dx.doi.org/10.1145/2330601.2330605

Stein, R., & Brennan, S. E. (2004). Another person's eye gaze as a cue in solving programming problems. *Proceedings of the 6th International Conference on Multimodal Interfaces* (ICMI '04), 13–15 October 2004, State College, PA, USA (pp. 9–15). New York: ACM. http://dx.doi.org/10.1145/1027933.1027936

Tzafilkou, K., & Protogeros, N. (2017). Diagnosing user perception and acceptance using eye tracking in web-based end-user development. *Computers in Human Behaviour*, *72*, 23–37. http://dx.doi.org/10.1016/j.chb.2017.02.035

Verbert, K., Govaerts, S., Duval, E., Santos, J. L., Van Assche, F., Parra, G., & Klerkx, J. (2014). Learning dashboards: An overview and future research opportunities. *Personal and Ubiquitous Computing*, *18*(6), 1499–1514. http://dx.doi.org/10.1007/s00779-013-0751-2

Vohs, K. D., & Baumeister, R. F. (2016). *Handbook of self-regulation: Research, theory, and applications*. New York: Guilford Publications.

Watson, C., Li, F. W. B., & Godwin, J. L. (2013). Predicting performance in an introductory programming course by logging and analyzing student programming behaviour. *Proceedings of the 13th International Conference on Advanced Learning Technologies* (ICALT 2013), 15–18 July 2013, Beijing, China (pp. 319–323). IEEE Computer Society. http://dx.doi.org/10.1109/ICALT.2013.99

Welch, B. L. (1951). On the comparison of several mean values: An alternative approach. *Biometrika*, *38*(3/4), 330–336. http://dx.doi.org/10.2307/2332579

Winne, P. H. (1982). Minimizing the black box problem to enhance the validity of theories about instructional effects. *Instructional Science*, *11*(1), 13–28. http://dx.doi.org/10.1007/BF00120978

Winne, P. H. (2006). How software technologies can improve research on learning and bolster school reform. *Educational Psychologist*, *41*(1), 5–17. http://dx.doi.org/10.1207/s15326985ep4101_3

Wise, A. F. (2014). Designing pedagogical interventions to support student use of learning analytics. *Proceedings of the 4th International Conference on Learning Analytics and Knowledge* (LAK '14), 24–28 March 2014, Indianapolis, IN, USA (pp. 203–211). New York: ACM. http://dx.doi.org/10.1145/2567574.2567588

Yoon, D., & Narayanan, N. H. (2004). Mental imagery in problem solving: An eye tracking study. *Proceedings of the 2004 Symposium on Eye Tracking Research and Applications* (ETRA '04), 22–24 March 2004, San Antonio, TX, USA (pp. 77–84). New York: ACM. http://dx.doi.org/ 10.1145/968363.968382