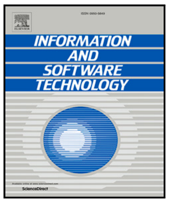
Information and Software Technology 174 (2024) 107502

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



On current limitations of online eye-tracking to study the visual processing of source code

Eva Thilderkvist, Felix Dobslaw [*]

Department of Computers and Systems Sciences, Mid Sweden University, Akademigatan 1, Ostersund, Jamtland, Sweden

ARTICLE INFO

Keywords:
Eye-tracking
Code comprehension
Webcam
Online experiment
Fixation algorithm

ABSTRACT

Context: Eye-tracking is an increasingly popular instrument to study how programmers process and comprehend source code. While most studies are conducted in controlled environments with lab-grade hardware, it would be desirable to simplify and scale participation in experiments for users sitting remotely, leveraging home equipment.

Objective: This study investigates the possibility of performing eye-tracking studies remotely using open-source algorithms and consumer-grade webcams. It establishes the technology's current limitations and evaluates the quality of the data collected by it. We conclude by recommending ways forward to address the shortcomings and make remote code-reading studies in support of eye-tracking feasible in the future.

Method: We gathered eye-gaze data remotely from 40 participants performing a code reading experiment on a purpose-built web application. The utilized eye-tracker worked client-side and used ridge regression to generate x- and y-coordinates in real-time predicting the participants' on-screen gaze points without the need to collect and save video footage. We processed and analysed the collected data according to common practices for isolating eye-movement events and deriving metrics used in software engineering eye-tracking studies. In response to the lack of an algorithm explicitly developed for detecting oculomotor fixation events in low-frequency webcam data, we also introduced a dispersion threshold algorithm for that purpose. The quality of the collected data was subsequently assessed to determine the adequacy and validity of the methodology for eye-tracking.

Results: The collected data was found to be of varying quality despite extensive calibration and graphical user guidance. We present our results highlighting both the negative and positive observations from which the community hopefully can learn. Both accuracy and precision were low and ultimately deemed insufficient for drawing valid conclusions in a high-precision empirical study. We nonetheless contribute to identifying critical limitations to be addressed in future research. Apart from the overall challenge of vastly diverse equipment, setup, and configuration, we found two main problems with the current webcam eye-tracking technology. The first was the absence of a validated algorithm to isolate fixations in low-frequency data, compromising the assurance of the accuracy of the data derived from it. The second problem was the lack of algorithmic support for head movements when predicting gaze location. Unsupervised participants do not always keep their heads still, even if instructed to do so. Consequently, we frequently observed spatial shifts that corrupted many collected datasets. Three encouraging observations resulted from the study. Even when shifted, gaze points were consistently dispersed in patterns resembling both the shape and size of the stimuli without extreme deviations. We could also distinguish recognizable reading patterns. Linearity was significantly different when participants were reading source code compared to natural text, and we could detect the expected left-to-right and top-to-bottom reading directions for participants reading natural text snippets.

Conclusion: The accuracy and precision levels were not sufficient for a word-by-word analysis of code reading but could be adequate for a broader, coarse-grained precision study. Additionally we identified two main issues compromising the collected data validity and contributed a fixation detection algorithm to approach one of these issues. With suitable solutions to the identified issues, remote eye-tracking studies with webcams on code reading could eventually be feasible.

* Corresponding author.
E-mail addresses: evth1400@student.miun.se (E. Thilderkvist), felix.dobslaw@miun.se (F. Dobslaw).

https://doi.org/10.1016/j.infsof.2024.107502
Received 27 February 2022; Received in revised form 30 May 2024; Accepted 31 May 2024
Available online 7 June 2024

1. Introduction

The use of eye-tracking in software education, research, and development has become increasingly popular in recent years [1–5]. Eye-tracking can be used to study code comprehension, identify source code complexity, and detect logical errors at review time. While it is a useful research tool offering insight into the cognitive processes behind programming tasks, it is also error-prone with many variables that can affect collected data quality. Key considerations include equipment, setup, execution environment, as well as participant positioning and physical attributes [6,7]. Most eye-tracking studies are performed in controlled laboratory environments, using special-purpose and often expensive eye-tracking equipment [1,8,9]. However, research in the field of cognitive psychology has shown that gaze data from simple fixation tasks gathered remotely with webcams from unsupervised test participants does not differ significantly from individuals performing the same experiment with webcams in-lab [10]. This revelation poses the question of whether rolling out experiments online would be a feasible option for scaling up eye-tracking studies. However, the development of webcam gaze-prediction technology is still very much an evolving field. Several open-source webcam gaze-prediction algorithms have been tested and found reasonably accurate [11,12], although not yet as accurate as laboratory eye-tracking practices.

This study examines the possibility of conducting an online eye-tracking study to gather gaze data specifically during the activity of reading computer source code and subsequently using that data to derive meaningful metrics for analysing the programmer's cognitive process. Webcam-eye-tracking technology would allow experiments using participants outside of the researcher's geographical area and also let participants use their own systems to collect the data.

Differentiating between eye-tracking reading behaviour and lower precision fixation tasks is essential because of its related challenges. Using a text stimulus rather than an image or layout design involves matching the gaze data to much smaller *Areas of Interest* (AOI), requiring a much higher level of accuracy and precision. Additionally, the requirement to systematically record data emerges when studying behaviours. Gaze-session data is commonly evaluated and presented with the aid of visual tools such as heatmaps [13–17]. These image overlays, depicting large concentrations of gaze or fixations with bright colours, are not always suitable for displaying reading gaze. A format that correctly accounts for chronological order is required for an accurate presentation of reading behaviour, also considering the time series of events. Further, the activity of reading code is very different to reading regular text, as it is a much less linear task. Thus, it is particularly important to differentiate those two, and here, focus mainly on the accuracy of reading source code, which poses a greater difficulty because of the many jumps programmers make when analysing and visually navigating a code.

In this study we set up an experiment to collect gaze data remotely with a client-side script on an online platform. We then evaluated the data to assess its quality and processed it in accordance with the steps commonly associated with deriving metrics for studying reading patterns. The motivation behind the study was the perception that eye-tracking is as dependent on the post-processing of data as it is on the functionality of the algorithms that collect it. Many webcam scripts are designed to track iris and gaze movement. However, there is insufficient evidence regarding their precision for empirical research. Due to the growing legal demands on privacy rights within ethical research, as well as the ambition to attract more participants, one objective was to process the eye-tracking data locally, in real-time so the tool could be used without invading any user's privacy (no video is ever transferred). Thus, we assess a practical implementation of a webcam eye-tracking study, evaluate the results, identify its shortcomings, and suggest improvements to overcome current deficits to see the technology evolve into a valuable research tool in the future. While we found that there were severe limitations in the technology early, we acknowledge that our initial intent was to investigate differences in eye-gaze between programming paradigms (imperative and reactive), which we had to accept as being too far a reach. This discovery though, led to this structural analysis of shortcomings, and one can, therefore, categorize this work, at least partly, as a negative result reporting study.

In the remote setting, varying data quality is expected as many factors cannot be controlled, such as the hardware running the script and the actions of the subject experimenting. Previous research with the same tool presents the limiting effect of the environment on the results [10]. However, in previous research, only low-precision tasks were examined, and in this study, working with smaller text AOI fragments while still having these uncontrolled environmental factors might have shown to have potentially larger negative implications on the outcome.

The cognitive reading process is closely tied to gaze-fixation events [18]. Fixations are oculomotor events during visual attention which can be extracted from a sequence of gaze locations with the help of eye-movement algorithms. Several of these algorithms exist [19,20], but we struggled to find one which could be successfully applied to data recorded at the frequencies we encountered. In response, we introduced a simplified dispersion threshold algorithm, which allowed us to proceed with our data processing and derive higher-level metrics. Despite the advancements in webcam eye-tracking, little work has been done to assess the validity of cognitive events identified from any collected data. Two pre-prints exist with the same or very similar method as mentioned here [21,22],[1] as well as another recently published study [23]. We formulated four closely related research questions to establish the effectiveness of webcam eye-tracking in the context of code reading:

RQ1. What quality of unprocessed data can be expected when eye-tracking remotely client-side with webcams?

RQ2. Can oculomotor fixation events be isolated from the raw data?

RQ3. Can fixations be organized into metrics capable of distinguishing reading patterns to differentiate between source code and text constructs?

RQ4. Is gaze data from an online webcam experiment sufficient for an eye-tracking study for the activity of reading source code, and if so, to what extent?

RQ1 investigates the raw data quality regarding sample frequency, accuracy, and precision. At the same time, RQ2 asks whether differentiable fixations needed to link the data to the cognitive processes can be extracted from the data. RQ3 then examines whether the fixations translate into different reading patterns using established code reading metrics. Finally, RQ4 addresses the overall questions of applicability for the approach. The contributions of this study are:

- An evaluation of a practical implementation of a code reading study using an open-source online webcam-eye-tracking algorithm,
- a fixation isolation approach for low-frequency datasets,
- a dataset[2] of online collected eye-gaze data, and
- identifications of the limits of the technology.

We start by giving an insight into eye-tracking technology in Section 2 and continue in Section 3 by exploring the state-of-the-art open source webcam-eye-tracking and the domain of source code eye-tracking. These are two techniques with little conjoint history, and we have based this work on pertinent information derived from both fields.

[1] Of which the first one is a pre-print of this article.
[2] https://github.com/EvaThil/EyesOnTheCode/tree/master/Dataset

In Section 4 we describe the detailed methodology used to conduct this study. Consisting of functions and procedures of the data-collection web app, as well as the extensive data analysis protocol. The study results are presented in Section 5, subdivided into sections corresponding to the different orders of processed data. Additional subsections also evaluate the practical experiment execution, and general visual observations. Finally, we close the paper by discussing the results in Section 6 and conclude our findings in Section 7.

2. Background

The earliest paper reporting eye-tracking to study program comprehension in 1990 [24] investigated the visual attention of university students reading binary search algorithms written in Pascal. Ever since there has been a steady incline in the interest in software-related scientific work [7], with a significant rise in publications seen after 2012 [1]. Eye-tracking, unlike more traditional code comprehension assessment methods such as think-aloud methodology/questionnaires [25–27], gives insight into the underlying cognitive process of the programmer [18] and does not distract or occupy time from the task at hand.

Eye-tracking is the general term used to record gaze points while a subject is watching stimuli or performing a task [28]. It works by locating the subject's pupils and mapping their positions to the corresponding on-screen gaze point through some form of calibration. To date, software research has applied eye-tracking with special-purpose eye-tracking cameras and performed in laboratory environments, often using headrests to stabilize head position [1]. In addition, single and multi-camera studies have been conducted, even in combination with other physiological measures such as EEG [8] and fMRI [29]. Modern eye-tracking cameras can record data with sampling rates ranging from 60 Hz to 2000 Hz and come in many different price ranges. The root cause of the lower expected accuracy in gaze assessment for web cameras can be traced to the operational differences as compared to specialized eye-tracking cameras. While eye-tracking cameras apply near-infrared light reflections to locate the pupils, webcam eye-tracking relies entirely on time-consuming image processing and facial recognition algorithms.

Gaze data is collected as stimuli relative x- and y-coordinates with associated timestamps. The unprocessed data collected from eye-tracking sessions is referred to as first-order data [30]. Further data processing is necessary to derive significant and usable information from the gaze location data by isolating second-order eye-movement events. In turn, fixation datasets can be used to calculate third- and fourth-order data metrics from which researchers can conclude the subjects' cognitive processes.

The quality and quantity of the first-order data are crucial for subsequent processing and, ultimately, the usability of the entire dataset [31]. Although there are no standardized measures for data quality, *accuracy*, *precision*, and *sample rate* are most commonly used [32,33]. Accuracy refers to the difference between true and measured gaze and can suggest the dimension of AOI the data could reliably be matched. Code and other text-based stimuli need a relatively high accuracy compared to images, as the AOIs often consist of rows or even single words. Precision refers to the dispersion of the gaze signal and is calculated as the root mean square (RMS) between consecutive gaze points. Poor precision is due to higher noise levels and may lead to incorrect classification of eye movements [31,34]. Lastly, sampling frequency restricts the quantity of recorded data. The eye is the fastest muscle in the human body. Small oculomotor events may get incorrectly identified or missed altogether if the eye-tracker cannot record at a high enough rate [33].

Gaze data can be classified into two main oculomotor events: *fixations* and *saccades* [35]. Fixations are stable gaze positions directly linked to the brain's cognitive processes [18], and saccades are rapid eye movements between fixations. Fixations hold the crucial information obtained from eye-tracking. These represent the periods when the brain actively absorbs what the eye observes. Additional events that are sometimes also of interest are micro-saccades and post-saccadic oscillations. As these are usually much shorter in duration and depend on high-quality data, we do not expect to be able to isolate them from data collected by webcams [36].

Numerous metrics have been proposed for quantifying information derived from eye-movement events in software research [30,37,38]. These are classified as third-order data if they concern quantifying fixations or saccades, or fourth-order data if they describe the sequence of these events. Examples of third-order data are *fixation count*, *fixation rate*, and *fixation duration*. Fixation count is the number of fixations either during a time span or in a specified AOI. Fixation rate is the total number of fixations in one AOI to another. Fixation duration is the combined or average duration of all fixations in an AOI or on a stimulus. Fourth-order data adds another dimension and describes the succession of fixations or saccades, also known as scan paths. Finally, *linearity* is a fourth-order data that describes the general reading order and search strategy.

3. Related work

Eye-tracking studies using webcams are relatively uncommon but attempted within several research fields, usually involving low-precision tasks or measurements. The technology was tested by individuals with severe speech and motor impairment (SSMI) as a cursor guide [39]. It was introduced as a method for interacting with a simple multiple-choice quiz application as an alternative to operating a mouse or a joystick. The experiment compared the webcam-operated cursor to one operated by an eye-tracking camera and found that subjects submitted answers considerably faster with the eye-tracking camera. The cause was attributed to latency in image processing by the webcam algorithm and the increased saccadic movements associated with SSMI, making it harder for the real-time webcam gaze point to stabilize. Another study compared webcam eye-tracking to eye-camera eye-tracking when assessing visual memory [40]. Subjects were presented with a sequence of paired visual stimuli containing some occasional stimuli repetitions. The eye-trackers recorded time spent looking at the different stimuli to identify recognition patterns and were both found to be adequate data collection methods. Neither of these studies involved matching the gaze position to an AOI smaller than a quarter of the screen size. A very recent pre-print by Ribeiro et al. [22] utilized similar fixation techniques as in this paper.

A few webcam eye trackers have been developed for online use. Two of these have been implemented and tested in online experimental settings [11,12]. TurkerGaze was created for crowdsourcing on Amazon Mechanical Turk [11]. The algorithm was evaluated in-lab simultaneously with a commercial eye-tracking camera on three subjects using a headrest and found to have a comparable accuracy of 1.32°. Next, it was applied in an online experiment consisting of two simplified games which incorporated calibration before validation as a part of the gameplay. Validation matched gaze to AOIs roughly 1/16 of screen size before proceeding to an image free-viewing task for evaluation where the user was asked to indicate the presence of images previously encountered in the task. Dubey et al. [41] applied eye-tracking to isolate sentences on Wikipedia.

WebGazer.js is a JavaScript library, easily incorporated into most websites [12]. It provides real-time gaze predictions without a separate calibration step. Instead, it relies on user interaction to train the built-in ridge-regression prediction algorithm. The script was reported to have a mean accuracy of 4.17° based on online collected data. In a cognitive psychology investigation on online data conduction [10], the accuracy was calculated to a mean of 18% relative to screen size. While the self-reported accuracy of the two algorithms suggests a better performance from TurkerGaze, the results are not comparable as they were measured in two very different settings. TurkerGaze measured accuracy in a controlled laboratory environment with a headrest to stabilize the subject. In contrast, WebGazer.js measured its accuracy from remotely sourced data with no control over participants's actions or positioning.

Diaz-Tula and Morimoto present a real-time fixation algorithm that uses a dispersion threshold similar to the one used in this paper for a range 30–60 Hz [42]. In this paper, we studied a range of about 22 Hz, and processing was not required in real-time but done in post-processing.

Several third- and fourth-order data metrics have been used in source code studies, albeit with infrared eye-tracking cameras [38]. Fixation count and duration have been used to identify the AOIs attracting the most attention to evaluate comprehension [24,43], to investigate the difficulty of finding errors [9,44], examine if particular task solving strategies are more efficient than others [45], and to determine if one programming language is easier to understand than another [46]. More fixations indicate that more effort is needed to process the code [27,37,47–49]. Higher fixation duration can separate less relevant AOIs from more relevant ones. Scan paths and linearity can reveal the programmer's approach to processing the source code. The scan path can indicate how effectively a bug can be found [9], and the path between fixations also indicates how the programmer's attention switches during a task. Lower switch rates are associated with higher programmer experience [48]. Also connected to the programmer's experience is linearity revealing different search strategies when visually processing the code. Reading source code has been proven to be less linear than reading natural text, and a greater programming experience has been connected to even lower linearity [5,50,51].

An alternative approach to track visual attention has been suggested where the camera element is excluded altogether. A Restricted Focus Viewer (RFV) allows the recording of visual focus by blurring the code and allowing the programmer to display a clear, readable area by positioning the mouse over it [52]. REyeker is a tool recently presented for this purpose. It was designed to facilitate remote eye-tracking of source code to give the researcher a basic understanding of the developer's reading behaviour [53]. The design of such tools assures relatively high spatial accuracy, but it has been argued that adding an RFV will cause the reader to approach the task unnaturally. Furthermore, the number of attention switches is significantly reduced when using the RFV, and the average fixation duration increases, indicating a higher cognitive workload. In another study, Bednarik and Tukiainen found that using an RFV interfere more with the natural behaviour of experienced programmers who have to make greater adjustments to their natural code processing strategies while using it [54]. Vos et al. in [23] further demonstrate the effectiveness of webcam-based eye tracking in the Visual World Paradigm, which involves monitoring where people look as they listen to spoken sentences and relate them to visually presented information. This typically requires less spatial precision than tasks like reading detailed text or code. The focus is more on the general area of interest (distribution of look) rather than the precise tracking of eye movements across individual lines of text. This might be less sensitive to the lower spatial resolution or higher noise levels associated with webcam-based eye tracking.

4. Research methodology

We invited computer programmers to partake in a reading experiment performed in its entirety on an online platform.[3] The experiment had the structure of a quiz where the participants were shown short snippets of either natural text or source code and had to answer a simple question to confirm that they had read the text and understood the code. The data collected included: participant demographics, participant system details, validation gaze data, and reading gaze data.

Three levels of stimuli complexity were included to establish whether it was possible to isolate different reading patterns based on the composition of the text. Natural text snippets were compared to imperative Java source code to determine if two considerably different text styles could be differentiated. The imperative code paradigm was also compared with the reactive paradigm to determine if variations within different source code structures could be detected. The assumption was that code is very different from natural text, but that the code from different paradigms is still more similar to one another as it is still code. Reactive programming has gained popularity recently due to its declarative style extensions offering stream-oriented development.[4] Moreover, it has been argued that reactive programming is easier to comprehend, although there is not much empirical evidence on that end so far.

4.1. Eye-tracking algorithm and calibration

To record the eye-movements we selected the open source JavaScript eye-tracker WebGazer.js.[5] The script executes client-side and uses facial markers and ridge regression machine learning to produce screen relative x- and y-coordinates with associated time-stamps. The script has been reported to have moderate accuracy [10,12,39], but was chosen for three compelling reasons. The first reason was its continuous support and improvements contributed by an active GitHub community. The script version tested in previous studies is a few years behind the utilized implementation, and two notable improvements have been introduced. These were the changes to a more accurate face detection algorithm and the introduction of a threaded regression model. The second reason for choosing a web script, and central to the purpose of this study, was its practicality when setting up a remote-only experiment. Finally, the gaze algorithm was executed from the client's side with the hope of attracting more participants. Because no video had to be collected, the experiment can be considered less invasive to a participant's privacy [55].

To fit the purpose of studying source code, we had to modify the script to be used "out of the box" by removing the automatic mouse-interaction-based calibration and introducing a separate calibration procedure. This modification was necessary because relying on mouse clicks and movements was considered unsuited for the comparatively passive reading activity. Initial tests also indicated that cursor sampling distorted the gaze signal when the participant was not actively following the reading position with the mouse, a behaviour only some participants exhibited. We created a calibration method prompting the participant to look at several calibration points, appearing in random order but in fixed positions on the screen, one at a time (see Fig. 1).

Ten calibration samples were recorded 100 ms apart for each point and fed into the algorithm to train the ridge-regression model. This was repeated until the whole viewport was covered. A full calibration consisted of 25 points, while a half calibration, used as a refresher between every three snippets, contained only 16 points. The new calibration method was relatively elaborate compared to other eye-tracking implementations, but it was motivated by the desire to match predicted gaze to smaller AOIs.

Another difficulty encountered was the considerable lag between video input and face detection in the continuous asynchronous loop of the utilized iteration of the WebGazer.js script. This lag caused the browser's call stack to fill up, considerably slow down, and eventually crash. To remedy the lag, we introduced frequent pausing of the algorithm in the background to clear the stack. The script was paused at every point in the execution flow where active gaze data collection was unnecessary. We observed a considerable performance boost even when pausing for a single second between displaying calibration points.

[3] https://github.com/EvaThil/EyesOnTheCode/tree/master/ReadGazeStudy

[4] see, e.g., https://reactivex.io/.

[5] https://webgazer.cs.brown.edu/

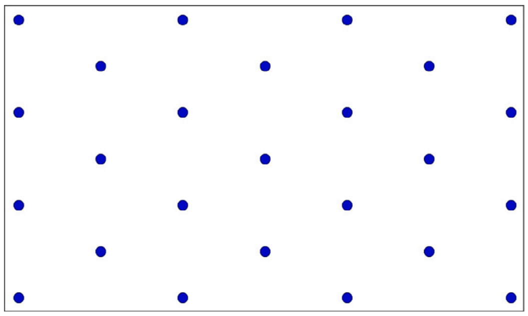


Fig. 1. Calibration point coverage relative to the viewport.

Table 1
Post recruitment filtration.

	Analysed for	Participants
Total participants		40 (100%)
Passed calibration	First-order data	34 (85%)
Passed data thresholds	Second-order data	32 (80%)
Competent programmers	Third- & fourth-order data	25 (63%)

4.2. Participants

Programmers were invited using convenience sampling to partake in the experiment by receiving the link to the online platform. The only precondition for participation was for the participant to be familiar with source code and programming concepts. The somewhat relaxed approach to the recruitment process was adopted to attract more participants as the first- and second-order data analysis was only investigating data quality and, therefore, independent of the participants's programming skills. Instead, a post-experiment filtration process to exclude ineligible participants from the skill-dependent tasks was implemented by evaluating the quiz answers (see Table 1).

The link was primarily distributed among current and former software engineering students at Mid Sweden University and Lund University. It was also in a limited capacity circulated within the computer departments of two automation companies based in Lund and Malmö, and a Facebook group for women in technology. In total, 40 participants started the experiment. Eighteen recruits were students, eight working professionals, and fourteen joined through the Facebook campaign.

Two students, one professional, and three social media recruits did not pass the first calibration. The remaining 34 participants were included when analysing first-order data, as they had all completed at least three snippets and one validation session. Another two participants were excluded before second-order data processing as we considered them to have insufficient sampling frequencies (<10 Hz). 32 participants remained for the fixation data analysis, of which 27 were fully completed experiments. A benchmark of at least three correctly answered source code snippets was set for a participant to be considered "programming competent". Only competent programmers were considered when evaluating third- and fourth-order metrics. Only ten (one female) competent participants declared a familiarity with reactive programming and were included in the paradigm comparison. Table 2 shows the distribution of age, gender, and self-reported Java experience for the 25 competent participants.

4.3. Experiment design

The website hosting the experiment was constructed specifically for the study. Participants were greeted with a short presentation of the experiment before being redirected to a questionnaire to collect demographic data. In addition to the questionnaire data, technical information about the client system, easily obtainable by the browser was automatically collected. Before starting, the participants were shown an

Table 2
Age, gender and experience demographics of the 25 competent Java programmers.

Male	16 (64%)
Female	8 (32%)
Other	1 (4%)
Student	11 (44%)
Hobbyist	7 (28%)
Professional programmer	5 (20%)
No experience	2 (8%)
Age in years	36 ± 9

example snippet for practice. The purpose of the practice snippet was to allow the participant to familiarize themself with the quiz procedure and the expected stimuli format. To prepare for gaze data collection, the participants received instructions to ensure the eye-tracking script could obtain the best result [10], see Fig. 2, and also to close all other programs and tabs on their computer.

The gaze algorithm was calibrated and validated five times. At the start, a full calibration followed by a validation was performed. After every three snippets, participants were asked to do a shorter calibration followed by the same validation procedure. The validation sessions consisted of seven validation points shown directly after the calibration points used to calculate accuracy and precision. The validation points were integrated so that the participant was unaware of the transition from calibration points to avoid the Hawthorne effect, where the participant changes their behaviour as a reaction to feeling evaluated and thus getting a misrepresenting value. It is not uncommon in eye-gaze studies for increased awareness of a situation to cause a subject to "try harder" and thus display an unnatural behaviour [6]. The seven validation points were arranged in an H-shape covering the approximated boundaries of the snippets display area, see Fig. 3.

The gaze was recorded as a continuous data stream during the one-second display of each validation point. We established a metric, referred to as *fixation accuracy*, to obtain real-time feedback on the data collected during the experiment. We assumed participants correctly looked at the calibration point for the displayed duration and classified it as a fixation. The fixation accuracy was calculated as the Euclidean distance from the average coordinates of all the gaze points to the validation point. We expected a noisy gaze signal, this way we could disregard the flicker which we would later be addressing by filtering in subsequent processing steps. Eq. (1) shows the formula for fixation accuracy (ACC-FIX) where k denotes the validation point, and i iterates the individually recorded gaze points.

$$\text{ACC-FIX} = \sqrt{\left(x_k - \frac{1}{n}\sum_{i=1}^{n} x_i\right)^2 + \left(y_k - \frac{1}{n}\sum_{i=1}^{n} y_i\right)^2} \tag{1}$$

The final fixation accuracy was derived from the mean accuracy of all seven points in the validation session. A higher value indicates a greater offset and a lower accuracy. A calibration was considered unsuccessful if the fixation accuracy value was larger than a third of the stimuli height. This benchmark was generous, but results showed participants either cleared it with a good margin or grossly exceeded it. If exceeded, a participant was redirected to do a new full calibration. After four unsuccessful consecutive calibrations, they were informed they most likely had an unsuitable setup and could not continue the experiment. Because of the assumption that the participant was correctly fixating on the validation point as instructed, this accuracy formula was considered more appropriate for practical real-time application than measuring each Euclidean distance from individual gaze points to validation point as done in previous studies as we would otherwise likely capturing a lot of noise [10,12].

When performing the reading exercise, each participant was presented with twelve text or code snippets while the webcam and script ran continuously until an answer was submitted. Fig. 4 displays such

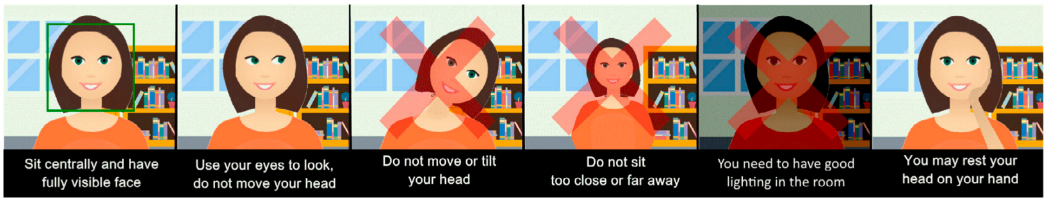

Sit centrally and have fully visible face
Use your eyes to look, do not move your head
Do not move or tilt your head
Do not sit too close or far away
You need to have good lighting in the room
You may rest your head on your hand

Fig. 2. The instructions given to participants before commencing the experiment. The second image instructing the participant to maintain her head position remained visible for the duration of the experiment.

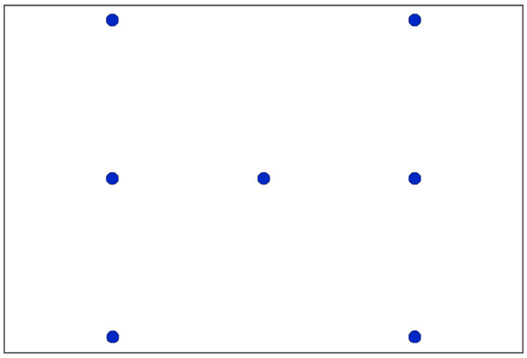
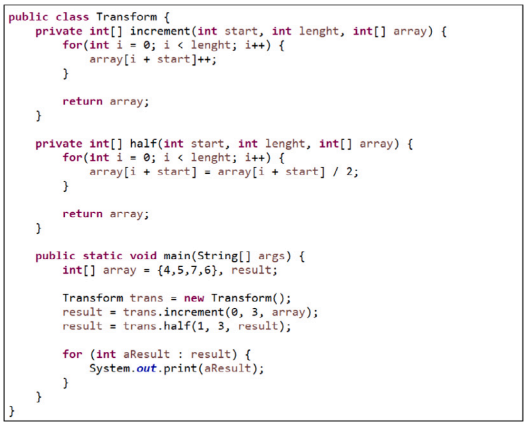


Fig. 3. Validation points covering approximated snippet display area.



```java
public class Transform {
    private int[] increment(int start, int lenght, int[] array) {
        for(int i = 0; i < lenght; i++) {
            array[i + start]++;
        }

        return array;
    }

    private int[] half(int start, int lenght, int[] array) {
        for(int i = 0; i < lenght; i++) {
            array[i + start] = array[i + start] / 2;
        }

        return array;
    }

    public static void main(String[] args) {
        int[] array = {4,5,7,6}, result;

        Transform trans = new Transform();
        result = trans.increment(0, 3, array);
        result = trans.half(1, 3, result);

        for (int aResult : result) {
            System.out.print(aResult);
        }
    }
}
```

Fig. 4. An example of imperative Java code for the reading exercise (snippet 10).

an example. The web app collected the generated eye-gaze coordinates, stored them locally, and sent them to the server after each completed snippet. It was done both to secure the data in case of malfunction and to free up browser memory to maximize the script's performance. To validate comprehension, the same question was given for every source code snippet: *What is the output of the* ***System.out.print()*** *statements?* For the natural text snippets, an easy question to identify a detail in the text was asked instead. The purpose of having the participant submit answers was primarily to confirm that they had read and understood the code or text as opposed to measure the level of programming expertise in any greater detail. Therefore, some lenience was given when correcting the answers. Simple misunderstandings of specific operators were also accepted, as were minor misspellings. An example is the reactive factory *interval* which emits on zero, was sometimes confused with starting on one. Formatting mistakes such as forgetting trailing zero on emitted double were also accepted. A snippet bank[6] with fifteen code snippets of each paradigm, and fifteen short story extracts were used. Each participant was given equal parts of each snippet type, chosen randomly but displayed alternating. Although the imperative source code snippets were written in Java, they displayed fundamental coding concepts to increase the chance of being recognizable by the wider coding community. RxJava snippets, representing the reactive programming paradigm, were only given to those participants previously stated having a familiarity with the paradigm. All code snippets were constructed with the syntax of educational examples [56–58]. The story extracts used in the natural text snippets were taken from fairy tales compiled by the Brothers Grimm [59].

Apart from the gaze data, the resolution and position of the stimulus viewport concerning the entire client screen were also collected for each completed snippet. This information allowed us to normalize the data as we had no way of controlling the screen resolution of participants' systems.

The web app interface design was essential to maintain the experiment's integrity. The participants were given short but clear instructions not to overwhelm or distract them. We further required minimal interactions with the website to reduce the chance of human error. Forward navigation and answer submissions were the only available actions. All data collection, apart from the questionnaire and snippet answers, was automatic.

After completing the experiment, participants were shown their achieved score and given an opportunity to submit feedback about their experience of the experiment. They could also volunteer a single video frame from the webcam feed to offer us a hint at their camera quality and seating position. To protect the participants privacy and not discourage them, a guarantee not to publish these images was given.

4.4. Data processing and analysis

We processed and analysed the collected data following steps associated with deriving metrics used in software eye-tracking studies [30]. We separated the gaze data consisting of x- and y-coordinates and timestamps into two formats. Validation data consisting of gaze sessions corresponding to the individual validation points and snippet data divided into gaze sessions for each completed stimulus. The data was processed in steps shown in Fig. 5. A Java program was created to perform the different processing steps.[7] The dissimilarities between collected webcam data and eye-camera data, for which most existing analysis programs are written, created a necessity to add additional processing capabilities. Two reasons for that were vastly varying sampling rates and the many different screen resolutions at collection time.

In total, 34 datasets – one per participant – were used to assess the quality of the first-order data. We calculated both fixation accuracy as previously defined and *gaze accuracy* (ACC-GZE) [10,12,60] in pixels and degrees for each dataset. Gaze accuracy is the average Euclidean distance to the validation point from all the individual gaze points as defined by Eq. (2).

$$\text{ACC-GZE} = \frac{1}{n}\sum_{i=1}^{n}\sqrt{\left(x_k - x_i\right)^2 + \left(y_k - y_i\right)^2} \tag{2}$$

[6] https://github.com/EvaThil/EyesOnTheCode/tree/master/Stimuli

[7] https://github.com/EvaThil/EyesOnTheCode/tree/master/ReadDataAnalyser

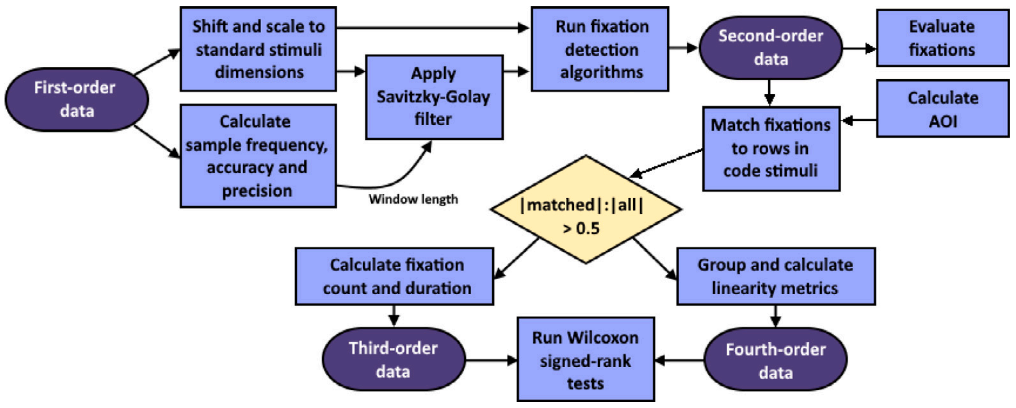

First-order data
Shift and scale to standard stimuli dimensions
Calculate sample frequency, accuracy and precision
Apply Savitzky-Golay filter
Window length
Run fixation detection algorithms
Second-order data
Evaluate fixations
Calculate AOI
Match fixations to rows in code stimuli
|matched|:|all| > 0.5
Calculate fixation count and duration
Group and calculate linearity metrics
Third-order data
Run Wilcoxon signed-rank tests
Fourth-order data

Fig. 5. Data processing steps.

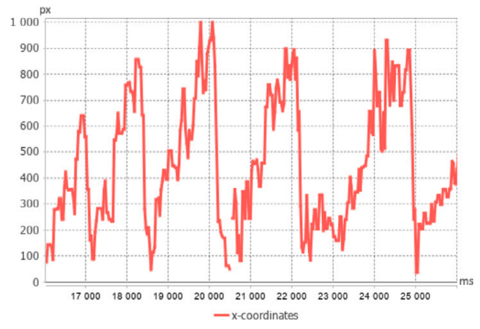


Fig. 6. Extract of unfiltered data from a participant reading natural-text snippet 10.

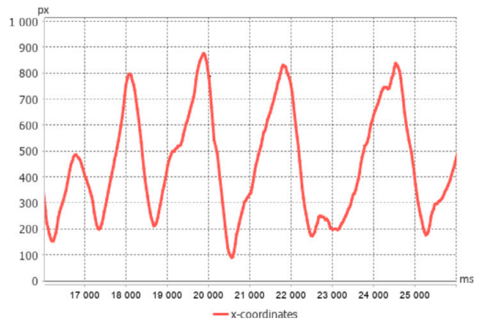


Fig. 7. The same data as Fig. 6 after filter applied.

A relative error margin was also calculated concerning the viewport's size. For the sake of simplicity, we approximated the participant to sit 50 cm away from the screen when calculating the visual angle, see also [61]. We calculated sample frequency by dividing the response time for each snippet session by the number of collected gaze-readings and then averaged over all of the participant's sessions. Finally, precision was calculated as the root mean square of the distance between consecutive gaze points (RMS-S2S) [32,60]. Because of the different sampling frequencies observed, we calculated precision on the validation data only, not the snippet data.

$$\text{RMS-S2S} = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n-1}\left(\left(x_{i+1}-x_i\right)^2+\left(y_{i+1}-y_i\right)^2\right)} \tag{3}$$

By calculating the precision of what was expected to be a stationary gaze, some of the variation caused by different sample frequencies could be minimized. Eq. (3) shows the formula for precision.

Collected raw gaze data coordinates were relative to every participant's screen resolution, resulting from being collected on various systems. The raw gaze data coordinates were also measured from the top left corner of the screen, not the relevant viewport where the stimuli were displayed. Before further analysis was carried out, the data had to be normalized and shifted to the standardized snippet dimension of 1000 × 800 pixels. Further data processing only continued with the 32 datasets, which passed the sampling frequency threshold of 10 Hz.

The gaze data contained a very high level of noise, as was indicated by high RMS-S2S values, and could be observed from the high spatial dispersion between samples, see Fig. 6. To smooth the data, we applied a Savitzky–Golay filter over a window length equal to the sampling frequency, with a polynomial of three, see Fig. 7. The filter size was selected based on the simplified assumption that the eye performs, on average, three saccades every second [62,63].

We attempted to isolate fixations in the dataset with three different fixation detection algorithms. Most existing algorithms for isolating fixations in gaze datasets have been developed to use on data collected from laboratory-grade eye-tracking cameras [19]. We evaluated two such algorithms. The first one was a velocity-based algorithm implemented in the R library "Saccades".[8] It detects fixations as positions where the gaze is stationary based on the change in the velocity between the gaze points. We applied it to the filtered dataset. The second algorithm used was the M12 algorithm. M12 is a 2-means clustering algorithm initially developed for gaze data collected from children and infants [20]. Much like webcam datasets, data from youth often have high noise levels and sometimes periods of data loss. The poor data quality is attributed to the natural tendency of children and infants to struggle to sit still. The M12 algorithm was applied to the unfiltered dataset. Because of the low sampling frequencies, the down-sampling function of that algorithm had to be switched off.

Few studies have investigated the accuracy of fixation detection algorithms on low sample rate webcam data. We failed to detect many fixations in frequencies below 20 Hz with any of the other algorithms. The absence of a suitable fixation isolation method compelled us to develop a simplified spatio-temporal dispersion threshold algorithm inspired by the dispersion-based algorithms used for some low-frequency eye-cameras [64,65] (see Algorithm 1). The algorithm defined a fixation as any position where the gaze lingered for over 150 ms [66], within the radius of one-row height (in the standardized snippet resolution that translates to 27px). The introduced *Recursive Longest* (RLG) algorithm works by sliding a window over the gaze vector (lines 6–15) and extending that window (lines 8–10) until the gaze data in the window no longer satisfies the spatial threshold for a fixation. When the threshold is reached, the violating gaze point is removed (line 11), and the fixation is temporarily saved (lines 12–14). Once the entire gaze vector is explored and the most prolonged fixation

[8] https://github.com/tmalsburg/saccades

exceeds the minimum time threshold (line 16), the fixation is added to the return vector, and the algorithm recursively calls itself again (line 19). To compare the output of all three fixation algorithms, we excluded fixations shorter than 150 ms from the other datasets. These short fixations could only be detected in the higher frequency data and prevented a fair comparison between all the collected datasets.

Algorithm 1 RLG

```
1:  procedure LONGEST(gazeVector) return fixationVector
2:      start ← start index of search window = 0
3:      end ← end index of search window = 1
4:      longestFix ← gaze vector, longest so far
5:      currentFix ← gaze vector, current fixation

6:      while end < gazeVector.SIZE do
7:          currentFix ← gazeVector[start++, end++]

8:          while all gaze in currentFix is within rowHeight
                    of central gaze point do
9:              currentFix.ADD(gazeVector[end++])
10:         end while
11:         currentFix.POP

12:         if currentFix.SIZE > longestFix.SIZE then
13:             longestFix = currentFix
14:         end if
15:     end while

16:     if longestFix.DURATION < 150ms then
17:         return []
18:     else
19:         return LONGEST(gazeVector[0, start -1])
                .ADD(FIXATION(longestFix))
                .ADD(LONGEST(gazeVector[end +1,
                    gazeVector.SIZE - 1]))
20:     end if
21: end procedure
```

Before being displayed on the experiment website, all the code and text snippets had been converted into identically formatted image stimuli. With the help of the original text files and the knowledge of the monospace font size used, the boundaries of the different text sections in the stimuli could be calculated. In the next step, all generated fixations were attempted to match AOIs defined as individual rows. If fixation was located between two AOIs, but still within a valid matching range (row height −1px), it matched the closest one. Finally, the three fixation algorithms were compared according to the number of fixations generated and the number of fixations matchable to stimulus rows. All gazes outside the bounds of the stimuli were excluded, e.g., participants looking at the answer textbox or outside the screen perimeter.

The RLG dispersion algorithm was selected for third- and fourth-order data analysis because it was the only one consistently generating a more considerable amount of fixations across different frequency datasets. We selected two third-order data metrics for measuring visual effort where greater values indicate higher cognitive load [38]. Fixation Count (FC) [27,37,47] was calculated as the number of fixations per word in the stimulus. Fixation duration was calculated as Average Fixation Duration (AFD) [54,67] within the whole stimulus.

The sequence/scan path of matched fixations for competent participants was grouped into linearity categories as initially defined by Busjahn et al. [50] (see Table 3). These metrics describe the reading behaviour defined by the direction of the saccades between fixations and are not mutually exclusive. A *vertical later* is any forward reading behaviour. Whether the gaze stays on the same line, moves down one line, or moves down several. A *vertical next* is a more precise forward movement, staying on the same line or progressing to the following immediate line. A *horizontal later* reads forwards on the same line. By this definition, a vertical next is also a vertical later, the same way a horizontal later is a vertical next and a vertical later. There are two types of regressive reading behaviour. The *regression rate* describes all regression, and *line regression* refers to the more specific behaviour of backward reading on the same line. Each metric corresponds to a percentage representing the ratio of all fixation switches belonging to that category and is averaged for each participant and snippet type.

We applied a Wilcoxon signed-rank test to each metric to compare the linearity of the natural text with the imperative code and the imperative code with the reactive. In addition, we also tested the null hypothesis for fixation count and fixation duration to establish if there was any significant difference between the sample populations. To reduce the risk of type-I errors, we applied Bonferroni correction, i.e. increased the threshold for significance by applying smaller p-values in correspondence to the number of metrics.

Finally, we performed a visual examination of the gaze data. Gaze data must often be visualized in context to be understood [6]. We plotted the fixations on top of the stimuli, observing both their positioning and onset over time. Heatmaps were also constructed to study the spatial dispersion and density of gaze.

5. Results

5.1. Experiment execution

Ten participants terminated their participation before the first calibration. Most of these explained they had done the practice snippet and decided they did not have the appropriate programming skills to continue. One participant failed to initialize his/her camera. Six participants started but failed the first calibration. All had a relative fixation error above 33%. Five of six could be attributed to the algorithm not starting correctly. Examination of the validation data revealed that no unique coordinate values were recorded, just the same coordinate repeated multiple times for an unknown reason. Two participants did not pass the second validation. Analysis of the validation data from these two participants showed sample frequencies of four and six Hz, insufficient for correctly determining the accuracy. The reason for the remaining six non-completed tests is undetermined. A total of 34 datasets were analysed, of which 27 were completed experiments containing data from twelve snippets.

Feedback collected at the end of the experiment reflected both positive and negative experiences. Some thought it was fun and exciting, but some expressed frustration with the length of the calibration procedure and/or the requirement to keep still. 25 out of 27 participants who completed the experiment agreed to send an image from the camera, suggesting that privacy was not an issue after all. Eight of these images revealed a seating position deemed too far away according to instructions. However, no parallels could be drawn between these participants and the quality of the data. Two participants had what was considered inappropriate lighting in the room. One had a large bright window in the background, but it did not affect the collected data. The other sat in a very dark environment. That dataset contained medium-quality data but with severe spatial distribution irregularities and was later excluded. It is unknown if this was due to the lack of lighting, the fact that the participant was wearing glasses, or if the participant had failed to remain still.

Examination of all the datasets revealed no significant adverse effects on the gaze prediction algorithm for participants wearing glasses.

Table 3
Metrics for linearity. F is the set of all recorded fixations. F_i (where $i = 1, \ldots, n$) is the fixation recorded at time index i. L(F_i) is the line number of the fixation at index i. In each trial, W is the set of word indices in the text. W(F_i) is the word number of the fixation at index i [50].

Measure	Definition	Computation
Vertical next	% of forward saccades that either stay on the same line or move one line down.	% of all F_i, where $L(F_i) - L(F_{i+1}) = \{0, -1\}$
Vertical later	% of forward saccades that either stay on the same line or move down any number of lines.	% of all F_i, where $L(F_i) \leq L(F_{i+1})$
Horizontal later	% of forward saccades within a line.	% of all F_i, where $L(F_i) = L(F_{i+1}) \wedge W(F_i) \leq W(F_{i+1})$
Regression rate	% of backward saccades of any length.	% of all F_i, where $W(F_i) > W(F_{i+1})$
Line regression rate	% of backward saccades within a line	% of all F_i, where $L(F_i) = L(F_{i+1}) \wedge W(F_i) > W(F_{i+1})$

Table 4
Sample rate in Hz grouped by core count.

	Qty	Sampling rate
2–4 cores	12	16.0 ± 8.12
6–8 cores	18	26.39 ± 9.6
10–12 cores	3	26.0 ± 7.0
14+ cores	1	36.0 ± 0
All systems	34	22.97 ± 10.12

Table 5
Script accuracy mean and standard deviation offset.

		Accuracy
Gaze	Pixels	152.17 ± 36.72px
	Visual angle	4.59 ± 1.1°
	Relative	15 ± 4%
Fixation	Pixels	105.68 ± 28.4px
	Visual angle	3.19 ± 0.85°
	Relative	11 ± 3%

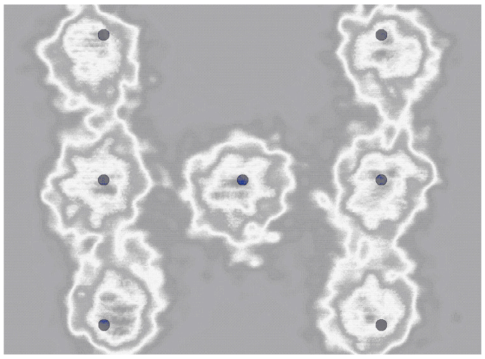


Fig. 8. Dispersion of all validation gaze.

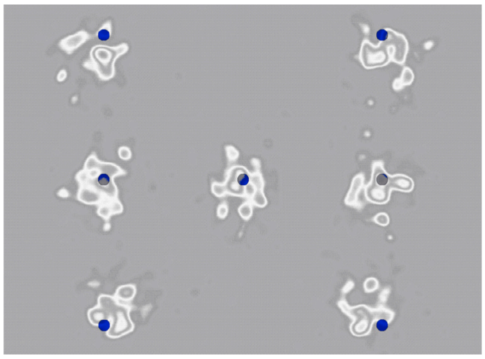


Fig. 9. Dispersion of normalized validation fixations.

5.2. First-order data - data quality

34 participants completed enough of the experiment to participate in the data quality analysis, with an average sampling frequency of 23 Hz (SD: 10). Client system core numbers varied from 2 to 16. In general, and as could be expected, a higher number of cores on the participant's system gave a higher sampling frequency, confirming the multi-threaded regression model's positive effect on the algorithm's efficiency (see Table 4). When we initially tested the script with the original single-threaded regression model, it never exceeded a sampling rate of 16 Hz, with most systems sampling under 10 Hz.

We measured first-order data for accuracy using two metrics: gaze accuracy and fixation accuracy. Table 5 displays the two metrics calculated in pixels, visual angle, and offset in percentage relative to viewport dimensions. For comparison, the relative height of the AOIs representing rows in the stimuli was 5.2%.

The gaze accuracy results are comparable to those from a previous in-lab study using the same script with a similar calibration procedure [10] while being roughly 3% better than the remote experiment in that same study. Because of the noisy nature of the gaze data, the fixation accuracy could be considered a better representation of the script accuracy, as filtration will consequently always be necessary. Assuming that the participant is correctly fixating on the stationary validation point, every gaze coordinate outside the gaze mean in recorded sampling frequencies could be considered noise. Fig. 8 shows the dispersion of validation gaze data from all participants, and Fig. 9 shows the same data as the average fixation point per validation point.

Precision hints at the spatial dispersion in the gaze signal. RMS is highly dependent on sampling frequency. To minimize this effect and make the collected datasets comparable, we only calculated precision on validation sessions which were expected to be stationary gaze points (see Table 6). Although arguably not a fair comparison, one could consider that a grossly approximated precision value for eye-tracking cameras is around 0.3° [32], which suggests a recorded noise level more than fifteen times larger for webcam signals.

Table 6
Gaze precision.

		Precision
RMS-S2S	Pixels	151.51px ± 45.03px
	Visual angle	4.66° ±1.39°
	Relative	16% ± 5%

We observed no significant trends between accuracy, precision and sample frequency in the recorded datasets.

5.3. Second-order data - fixation detection

The three fixation algorithms were evaluated by comparing the number of detected fixations within a fixed time interval, the number of fixations matchable to AOI as stimuli rows, and visually by plotting the fixations on top of the stimulus images. The returned number of

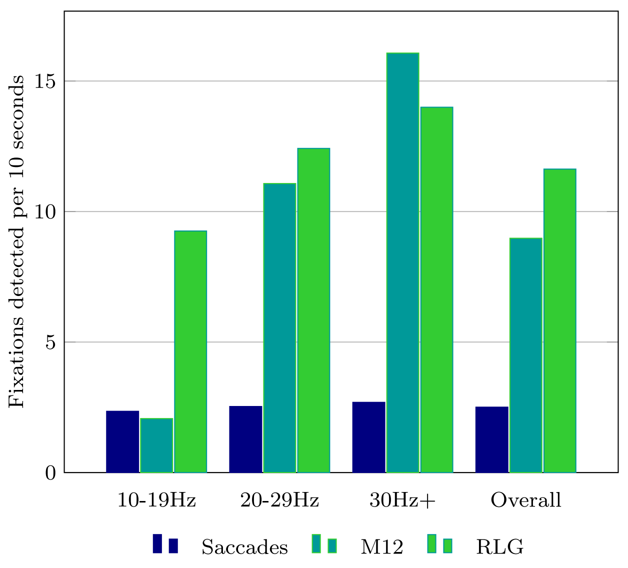


Fig. 10. Average number of fixations across all datasets generated by the different algorithms and grouped by sampling rate.

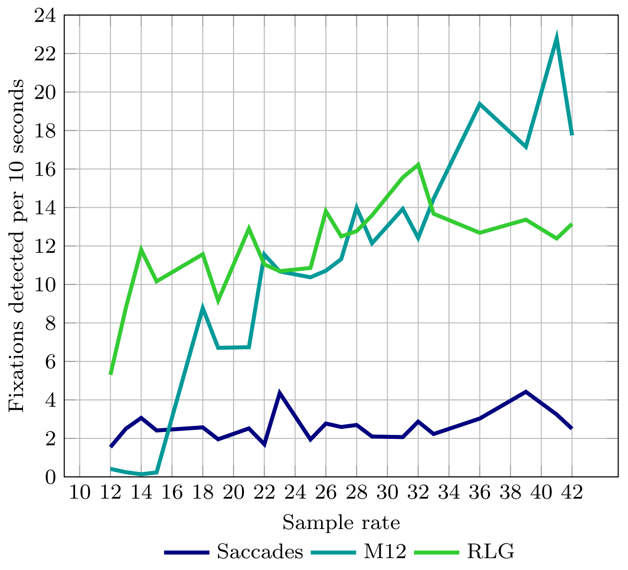


Fig. 11. Average number of fixations generated by the sample rate.

fixations for each algorithm grouped by sampling frequency is shown in Fig. 10 and broken down further in Fig. 11.

The velocity-based algorithm (Saccades) was insufficient for all the recorded frequencies. It was tested with a range of small and large velocity cut-off thresholds on both filtered and unfiltered data, with little difference. The M12 algorithm started working well at around 20 Hz and performed the best when the sample frequencies exceeded 30 Hz. M12 returned a reasonable number of fixations in high sample frequencies with a wide range of durations (see the top section of Fig. 12)

The majority of the datasets had a sampling rate below 30 Hz. The consistency of the RLG algorithm in generating many fixations on these datasets justified the decision to choose RLG to proceed with third- and fourth-order processing. When plotted on top of the stimuli, the fixation dispersions displayed the most logical patterns with RLG. The M12 would have given a more extensive range and a higher number of fixations, but using it further would have meant to exclude many datasets (48%). The M12 was successfully tested by its creators in noise levels up to 5.57°, concluding that the lack of sample data was most likely the single reason for the failure. The fixation numbers displayed in Fig. 10 are not directly comparable with the simplified assumption of three expected fixations per second. That assumption includes a more extensive range of fixations achievable by smaller eye movements that were not accurately detected within the recorded sampling frequencies. Still, the RLG algorithm did likely not return the total number of fixations as sampling frequency still influenced the results.

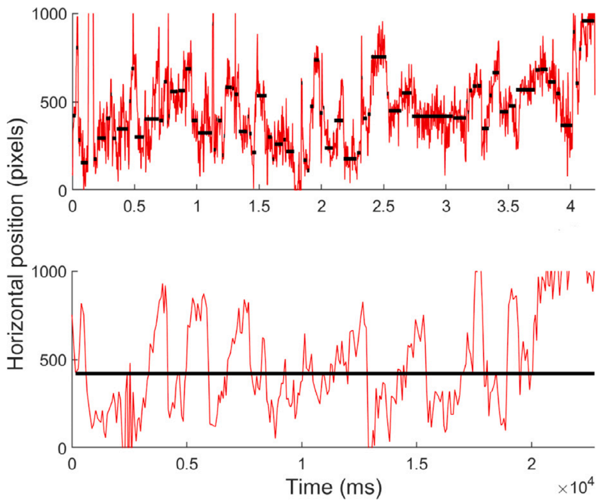


Fig. 12. Noisy data with high (top) and low (bottom) sample frequencies. M12 fixations are shown with black lines.

Table 7
Fixations within stimuli bounds matched to rows.

Algorithm	Match ratio
Saccades	70.66% ± 23.48
M12	74.38% ± 20.02
RLG	75.99% ± 20.07

The fixations generated by each algorithm were matched to corresponding rows of the stimuli to assess how spatially correct they were. The ratio of fixations matched to rows is presented in Table 7. A large part of the unmatched fixations can be attributed to a spatial shift present in some datasets. This trend is explored further in the result section describing visual observations.

When playing back the gaze data to observe the occurrence of the detected fixations in real-time, the M12 algorithm identified a series of randomly occurring fixations, probably because of the misidentification of signal noise. We understand this results from the algorithm's down-sampling function being switched off. The creators had set a minimum sampling threshold lower than that of the collected data frequencies, which caused it to throw an error if used. The misidentification of fixations in low-frequency data can be seen in the lower chart of Fig. 12, where the entire dataset is treated as one single fixation.

5.4. Third- and fourth-order data - identifying reading patterns

When comparing natural language text to imperative Java, the null hypothesis could be rejected for both third- and fourth-order metrics. One participant had to be excluded because of too many unmatched fixations (>50%), leaving 24 participants for this comparison. The two third-order metrics tested significantly differed when applied to the entire stimulus. Wilcoxon's signed-rank test for fixation count returned a Z-score of −4.271 and a p-score of 0.0, showing an apparent increase of fixations for Java code. The mean duration of fixations was also longer for the code with a Z-score of −3.700 and a p-score of 0.0.

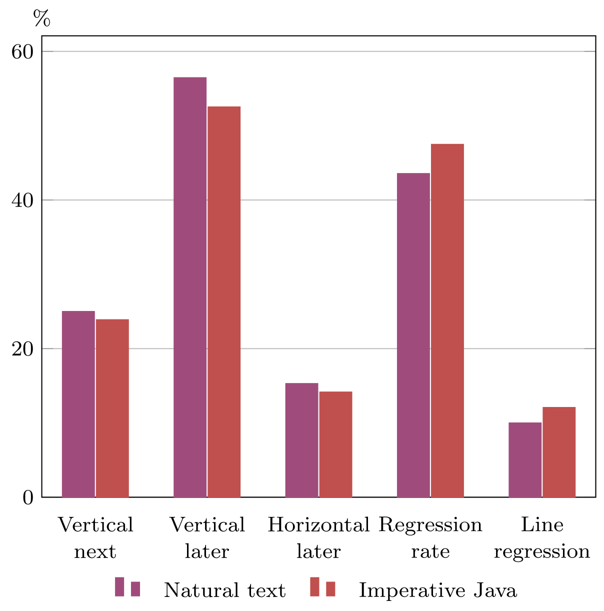


Fig. 13. Mean linearity metrics for natural text vs. imperative Java code.

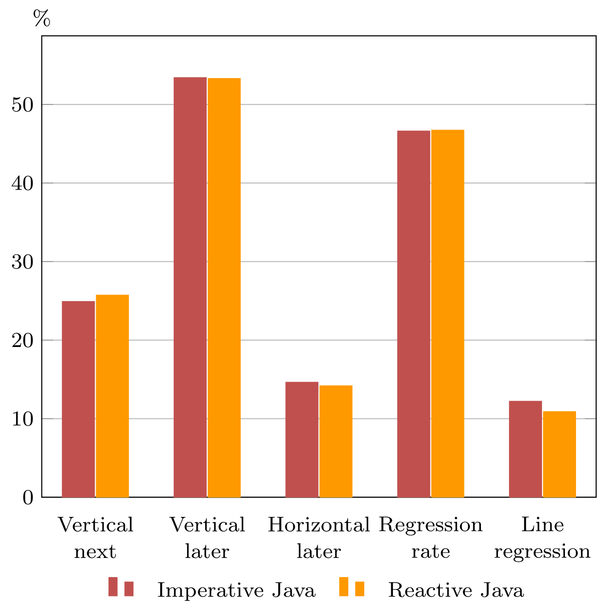


Fig. 14. Mean linearity metrics for imperative vs. reactive Java code.

Table 8
Wilcoxon signed-rank test for NT vs. Java. W is the sum of positive ranks.

Vertical Next	W = 187	Z = 1.043	p = 0.297
Vertical Later	W = 257	Z = 3.043	p = 0.002*
Horizontal Next	W = 177	Z = 0.757	p = 0.449
Regression Rate	W = 43	Z = −3.043	p = 0.002*
Line Regression	W = 80	Z = −1.986	p = 0.047

* Indicates distribution significance.

Table 9
Wilcoxon signed-rank test for imperative vs. reactive code. No significant difference could be found.

Vertical Next	W = 18	Z = −0.474	p = 0.636
Vertical Later	W = 23	Z = 0.0	p = 1.0
Horizontal Next	W = 23	Z = 0.0	p = 1.0
Regression Rate	W = 22	Z = 0.0	p = 1.0
Line Regression	W = 35	Z = 1.422	p = 0.155

The fourth-order linearity metrics are presented in Fig. 13. A significant difference could be found in two out of five metrics presented in Table 8. This rejection of the null hypothesis between code and natural text are in line with previous studies with eye-cameras [50,51]. However, those studies found significant differences in two additional linearity metrics, meaning the measurements from the webcam data could only discern the coarse vertical difference in reading strategy between the two text styles.

For the paradigm comparison, participants needed to know reactive programming, which reduced the number of available and usable (one participant had to be excluded — as above) datasets to nine. The only statistically significant difference in reading metrics detected between reactive and imperative code was an increased fixation count for the reactive code. Fixation count returned a Wilcoxon Z-score of −2.606 and a p-score of 0.009, suggesting higher visual effort was necessary to process the reactive paradigm code [27,37,47]. Fixation duration returned a Wilcoxon Z-score of −0.237 and a p-score of 0.813. No linearity metrics had any significant difference. The linearity results for the paradigm comparison can be observed in Fig. 14 and Table 9.

5.5. Visual observations

Although the gaze point offset for the gaze algorithm was relatively high compared to stimuli size and would suggest an insufficiency, there are useful observations within the datasets worth mentioning. When observing x-coordinate signals for participants reading natural text, a clear left–right pattern across row lengths could be observed. Likewise could a top-down direction in the y-coordinates. This corroborated the natural reading behaviour of such text. Some variation in the quality of these patterns was observed. Still, Fig. 15 displays an example for the x- and y-coordinates of a participant reading natural text snippet 10.

Gaze dispersions over the code stimuli were also observed to stay within the logical bounds of the code display area, shown in Fig. 16. This correctly scaled dataset confirmed an increased accuracy after filtration, suggesting accuracy levels are adequate for a valid study if used with larger AOIs.

The most significant visual observation was a spatial shift of gaze points in some datasets and, consequently, the derived fixations. The data appeared to be correctly dispersed but incorrectly mapped to the stimuli. The shifts presented more frequently on stimuli as the time from the last calibration increased, suggesting the calibration was wearing off. The most likely reason for this was the participant's head not being in the same position as when the calibration occurred. The shift was always down and sometimes slightly to the right, possibly indicating a slouching behaviour as the participant started relaxing. The algorithm does not have any support for offsetting head movements. Fixations from a shifted dataset are shown in Fig. 17, and the corresponding first-order gaze data in Fig. 18.

6. Discussion

6.1. Quality of recorded raw data

Unlike performing an eye-tracking study with one specific eye-tracking hardware setup, a range of different data-recording systems will produce a range of different quality datasets due to hardware variations in the client systems. The recorded datasets, despite different quality must both be processable and comparable to be used in the same context.

In response to RQ1, we interpret the data quality compared to special-purpose eye cameras. We measured significantly lower sample frequencies, higher noise levels, and less accuracy, but acceptable for tasks differentiating eye-gaze between larger areas on a screen.

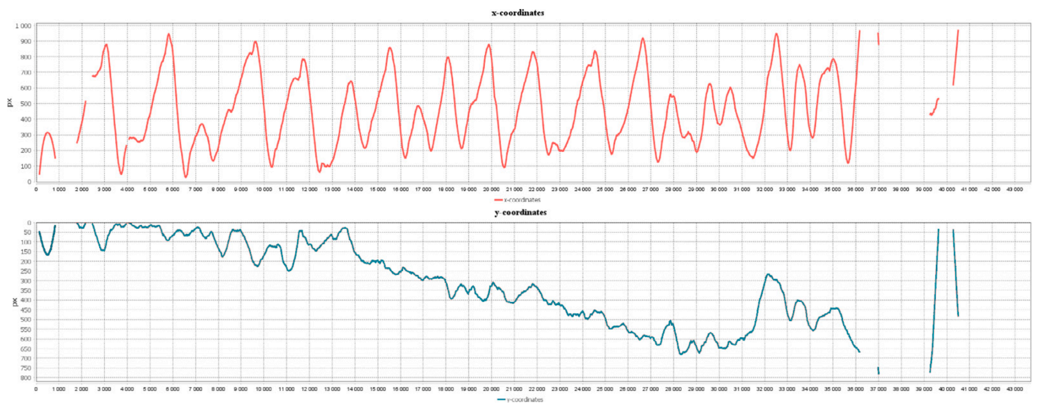
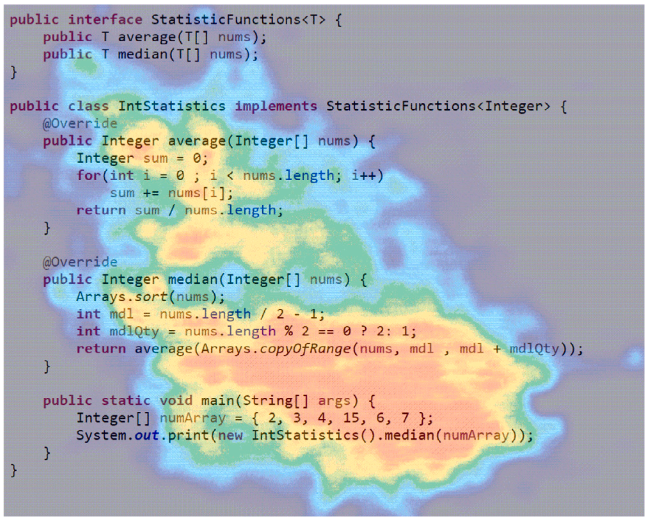


Fig. 15. Filtered x- and y-coordinates from a participant reading natural text snippet 10. Gaps in the gaze signal indicate visual attention outside of the viewport. E.g., looking at the keyboard.



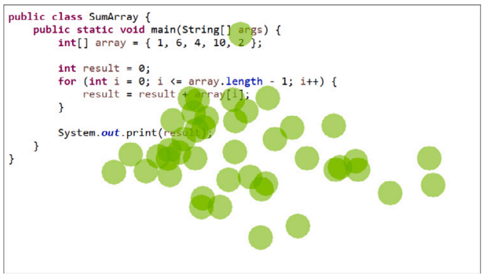
```
public interface StatisticFunctions<T> {
    public T average(T[] nums);
    public T median(T[] nums);
}

public class IntStatistics implements StatisticFunctions<Integer> {
    @Override
    public Integer average(Integer[] nums) {
        Integer sum = 0;
        for(int i = 0 ; i < nums.length; i++)
            sum += nums[i];
        return sum / nums.length;
    }

    @Override
    public Integer median(Integer[] nums) {
        Arrays.sort(nums);
        int mdl = nums.length / 2 - 1;
        int mdlQty = nums.length % 2 == 0 ? 2: 1;
        return average(Arrays.copyOfRange(nums, mdl , mdl + mdlQty));
    }

    public static void main(String[] args) {
        Integer[] numArray = { 2, 3, 4, 15, 6, 7 };
        System.out.print(new IntStatistics().median(numArray));
    }
}
```

Fig. 16. Heatmap of filtered gaze dispersion from seven participants.

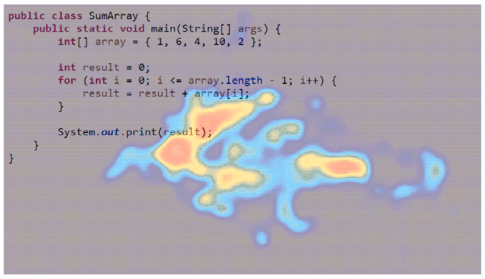


```
public class SumArray {
    public static void main(String[] args) {
        int[] array = { 1, 6, 4, 10, 2 };

        int result = 0;
        for (int i = 0; i <= array.length - 1; i++) {
            result = result + array[i];
        }

        System.out.print(result);
    }
}
```

Fig. 17. Fixations generated from a shifted dataset.



```
public class SumArray {
    public static void main(String[] args) {
        int[] array = { 1, 6, 4, 10, 2 };

        int result = 0;
        for (int i = 0; i <= array.length - 1; i++) {
            result = result + array[i];
        }

        System.out.print(result);
    }
}
```

Fig. 18. Shifted filtered first-order gaze data of Fig. 17.

The sampling frequency was only a fraction of that of an eye-tracking camera. The multi-threaded implementation of the eye-tracking algorithm improved sample frequency considerably. However, we observed that even powerful systems (many processors) could produce low sampling rates. Thus, minimum sampling rates cannot be guaranteed because the browser tab competes for resources with other tabs and processes on a client's system if the participant does not follow instructions. It is also worth mentioning that the sampling rate cannot exceed the frame rate of the available webcam. Current webcam technology is far from comparable with infrared eye-cameras' high sampling rates. There is room for improvement, though, as most modern webcams support 60 frames per second. Recording on more powerful systems and more efficient algorithms could also reduce computing time in the future.

Gaze accuracy for the webcams was almost ten times that of the 0.5° promised by most eye-tracking camera manufacturers, and so were noise levels. A close similarity was found between precision and accuracy values blurring the line between what was measured as noise and actual error in the data. This similarity confirms that the gaze data cannot be accurately used in its raw state. Filtration is necessary to decrease the flicker generated by the WebGazer.js algorithm. The filtered gaze signal shows all the suitable spatial characteristics when plotted on top of the stimuli and visually examined. However, extensive filtration also risks distorting the signal. High noise levels and low sampling rates have the most significant negative impact on the investigated eye-tracking technology. Together, they severely interfere with the efficiency of the fixation algorithms and consequently impede the likelihood of correct isolation of fixation events.

The spatial shifts encountered in the datasets were most likely the result of the calibration wearing off, i.e., the effect of participants having moved their heads. The shift we observed always occurred in the same direction, downwards and sometimes slightly right. The shift could be caused by a fault within the eye-tracking algorithm itself. Alternatively, the participants may have maintained a more alert posture while doing the calibration and then started to slouch as they relaxed afterwards while processing the snippets. We drew this conclusion from the observation that the gaze dispersions seemed to remain spatially consistent, even within shifted data. Therefore, we also assumed that accuracy values would improve if head movements were supported in the algorithm. In its current state, to be considered a valid eye-tracking method, the accuracy would have had to be twice as good as the recorded value to match the gaze to the height of the study's stimuli rows when unshifted. Defining individual rows as AOIs can be considered very specific, and many eye-tracking studies in software engineering do not use such high precision. Instead, general areas of code are defined as AOIs. If the integrity of the datasets were uncompromised by the shifts, an adequate level of accuracy would be achievable.

Due to the increased computational demand, there would also be a risk of decreasing sampling frequency associated with the introduction of head movement support when running on some systems.

RQ1: What quality of unprocessed data can be expected when eye-tracking remotely client-side with webcams?
The data quality will vary depending on the system used to record it, influenced by both the camera and the processor used. Accuracy and precision were found to be approximately ten times worse compared to an infrared eye-tracking camera, with sample rates measured between 12 Hz and 42 Hz, in contrast to the 60–2000 Hz range supported by infrared eye-tracking cameras.

6.2. Fixation isolation

Even though fixations can be obtained from the datasets, more work is needed to prove them to be accurately identified. To address RQ2, we proposed the RLG fixation algorithm, but the generated fixation dataset has not yet been validated against proven infrared eye-tracking cameras. Traditional eye-movement algorithms do not work correctly, mainly because of the low sampling frequencies and high noise levels. Velocity is ordinarily the most significant attribute of different eye movements and how to separate them. The raw data collected from the webcams were too noisy to detect noticeable velocity fluctuations. Extensive filtering was required to increase accuracy and precision, making the signal smooth and reducing the velocity fluctuations. Our spatio-temporal dispersion algorithm, RLG, indirectly used velocity by searching for the longest fixations first. A longer timespan encapsulating a lower spatial dispersion is the definition of low velocity in the gaze signal and, consequently, a fixation. The results of the RLG algorithm would have accurately plotted the path of the gaze but would not accurately determine fixation onset and duration. The low sample frequencies simply do not contain enough data [33]. We measured the number of fixations returned by the algorithms, with only limited ability to verify fixation algorithms. Also, an optimal algorithm would have generated a constant number of fixations between the different frequency datasets rather than show an increase for higher sampling rates. To correctly evaluate a fixation algorithm for webcam data, it should be compared to high-quality data fixations recorded in parallel or have pupils in the video feed manually inspected as a ground truth. More work must be done to correctly classify cognitive functions in low data qualities and quantities supporting cognitive psychology.

RQ2: Can oculomotor fixation events be isolated from the raw data?
We tested two established algorithms and failed to generate consistent results. We developed a dispersion-threshold algorithm that was successful at isolating a sufficient number of fixations. More work must be done to establish the validity of generated fixation datasets against an established ground truth.

6.3. Using the data for software engineering metrics

To answer RQ3, we derived some of the more commonly used metrics from the data while processing the stimuli as a whole text chunk without focusing on specific recognizable code attributes. A wide range of eye-tracking metrics to measure different cognitive processes exist in software engineering [30,38]. The purpose of the metrics chosen was to test the application, and it is possible that additional/different metrics could have been derived more successfully from the collected datasets.

Both third-order metrics, fixation count and duration, showed a clear distinction in reading behaviour between the natural text and the Java code. This distinction confirmed that reading source code takes more visual effort than natural text. Still, the source code could have exaggerated this by arguably having more difficult questions to answer.

The higher fixation count associated with reactive code in the paradigm comparison suggested the reactive code being read more thoroughly or possibly re-read. This extra effort could be explained by RxJava being a more specialized coding style and most participants being students with limited experience in reactive programming while working with imperative Java for several years.

Fourth-order data processing was done on what was already recognized as shifted datasets. Consequently, the results would not have accurately described the full extent of the linearity. Other studies have applied a manual adjustment protocol for datasets when shifts happen. Most datasets were shifted to start with, it was decided to leave them as they were for the sake of evaluation of applicability. Because no separation into smaller AOIs was done, and metrics were applied to the whole code stimulus, the shifts would have had a less significant negative impact than if matched to specific code identifiers only. However, these shifts would have had a more significant impact on reactive code snippets as they are naturally more spatially compact.

The linearity metrics could successfully distinguish code from natural text. This distinction was expected because of previously proven differences in reading patterns and considerable spatial dissimilarities in the text formats of the stimuli. However, the identified divergences were not identical to those in previous studies, and this could probably be attributed, in most part, to the shifted datasets and the use of non-identical stimuli. Furthermore, no distinction could be made between the two code paradigms. The data was probably too distorted, and the comparison was made on the entire code as one AOI instead of focusing on specific different code sections.

RQ3: Can fixations be organized into metrics capable of distinguishing reading patterns to differentiate between source code and text constructs?
We could differentiate between the source code and natural text reading patterns using our webcam generated fourth-order linearity metrics. We could not distinguish between the two code paradigms using the same metrics. The two webcam-generated third-order metrics, fixation count and fixation duration, also showed a significant difference between reading code and text. The only metric showing a significant difference between paradigm reading patterns was fixation count, where we detected a higher count for participants reading the reactive code, suggesting a higher effort was needed for this construct.

6.4. Suitability for source code eye-tracking studies

This section aims to answer RQ4. Given that our findings suggest clear limitations of the current technology, one could consider the study's outcome to be at least partially containing negative results, i.e. we failed in accurately localizing eye-gaze the precision required with the method and technology used.

In its current state, the webcam-eye-tracking technology falls somewhat short of the level of precision needed to study source code in detail. Accuracy and precision are insufficient to match gaze to areas as specific as single words reliably. Matching larger sections of code (in our case, more than two rows high) may be possible, but the spatial shifts caused by head movements compromise the validity of the results. Nevertheless, spatial distributions of the gaze signal show promising tendencies of remaining within the relevant ranges, suggesting gaze predictions are recorded in the correct scale.

Ultimately, because eye-tracking metrics are based on fixation and saccadic events, and not enough work has been done to validate methods of isolating them in webcam data, results cannot yet be verified. Nonetheless, we here list a summary of our findings as insufficiencies and promising observations.

Insufficiencies

- Limited sample frequencies
  - Unreliable fixation detection.

- Extremely high noise levels
  - Decreases accuracy as well as impedes fixation detection.
- Possibility of spatially shifted data.
  - Compromises integrity.
- Unverified eye-movement isolation algorithms
  - Raises questions of the validity of detected fixations.

Promising observations

- Accuracy good enough to track larger areas of code
  - Theoretically suitable for some computer-related eye-tracking studies.
- Consistent spatial distributions within the gaze signal
  - Maps screen coordinates to correct scale.
- Easy to deploy and participate
  - Accessible and economical.

RQ4: Is gaze data from an online webcam experiment sufficient for an eye-tracking study for the activity of reading source code, and if so to what extent?
The data is not of high enough quality for a high-precision study with a desire to match gaze with individual words or even lines. Coarse precision studies could be possible if interest was focused on code sections only and the AOIs were restricted to a minimum of 105px. The integrity of the data is also fully dependent on the subject sitting still throughout the experiment to avoid shifts in generated gaze points. Even if the right conditions are met, not enough research has been done to validate the data processing steps and, consequently, the accuracy of the data they generate.

6.5. Threats to validity

Internal. The validity of the data-processing results must be put into context because of the nature of the multi-step processing protocol. Although the study intended to perform all the steps necessary in an eye-tracking study, second-order data was dependent on the validity of first-order data, the same way third- and fourth-order data was dependent on the validity of the second-order. Any compromise in data would have subsequently compromised the data generated from it. More work is also needed to validate the correctness of both filter size and fixation algorithm.

We had to make certain assumptions regarding the remote participants, like the participant's seating position and ability to follow the instructions, actual conditions may have varied. Further, the choice to circumvent privacy issues by not collecting video to make the tool usable in most remote code-reading experiments has likely had an impact on the negative outcome of the study because of the parallel processing demands on the participants' computers. Further, in remote experiments, we have to assume that participants adhere to the set rules, increasing the risks of failure and negatively affecting the study's reproducibility.

The number of participants in the paradigm comparison was significantly smaller than in the natural text/code comparison due to the specific requirement of reactive programming competence, which could not always be satisfied. Furthermore, because of the online nature of the study, a larger group of participants was initially expected when constructing the experiment. The number of participants received was sufficient for first- and second-order data analysis. However, the bank of 45 questions might have been too large concerning the participant attendance to properly represent the different code types and compare datasets between participants. Hence variation was only compared within the same participant datasets and not between participants.

External. Stories chosen for the natural text stimuli were written in old English. This possibly increased the perceived text complexity for the participants who mostly had Swedish as their first language. The construction of the stimuli code also introduced a bias because it represented the author's perception of the two programming paradigms. Although the constructed code examples were syntactically correct and executable, they might not cover all aspects of the paradigms perceived by other programming community members. The decision not to reuse stimuli or data from existing studies was driven by our initial intent to study reactive programming in a high-level language. No suitable existing reactive programming study was available.

The size of the stimuli varied between participants because of screen resolution but was also affected by the original size set by the authors. The stimuli might not always have been displayed on the client system in a dimension representing a normal coding environment, such as in an IDE.

Finally, only a limited set of webcam eye-tracking algorithms are available, and there is no evidence that they differ considerably in quality and efficiency. Therefore, this study presents the results generated using one specific algorithm, and minor variations are expected to be obtained when applying others.

6.6. Implications and future work

We identified three areas where advancements would benefit webcam eye-tracking considerably and where we plan to focus future work.

All participants in these kinds of eye-tracking studies are unsupervised. Without an eye-tracking algorithm able to offset head movements, there is little hope of maintaining the integrity of the recorded gaze data. An eye-tracking algorithm must be developed to use additional reference points of specific facial features to determine if the head position has changed since calibration.

The WebGazer.js algorithm used in this study was developed for real-time applications. There is no need to display a real-time gaze-pointer to participants in an eye-tracking study. Cutting out the computation needed for this feature could save resources. We propose developing a webcam eye-tracking tool explicitly dedicated to eye-tracking studies. To preserve the participants' privacy, the data collected could only be the reference points of the pupils, facial features, and screen calibration points. Sampling rates should improve by focusing on collecting that data in real-time and leaving the prediction calculations to data processing time. This separation would also mitigate the penalty of head normalization calculations.

Finally, the recorded data will never be more valuable than the oculomotor movements generated from it. Work must be done to validate generated fixations from webcam data to develop an algorithm adequate for the task. Simultaneously recording an eye-tracking session with a webcam and a professional eye camera could generate data highlighting shortcomings and suggest solutions.

7. Conclusion

Eye-tracking is an increasingly popular instrument to study how programmers process and comprehend source code. While most eye-tracking studies are conducted in controlled environments with lab-grade hardware, remotely, leveraging home equipment would be desirable to simplify and scale participation in experiments.

Therefore, we tracked 40 participants in a semi-controlled code reading experiment on a purpose-built web application using open-source algorithms and consumer-grade webcams to better understand the obtainable quality of eye-gaze readings.

The webcam-eye-tracking method was ultimately proven insufficient for studying code reading in detail. Even though it is a negative result, this could provide valuable lessons to the community. Indeed, a deeper analysis presented in this paper revealed that the failure could

Table 10
Recommendations for researchers considering using webcam eye-tracking to study source code.

Recommendation	Expectation	Benefit
Restrict to participant systems with 6 or more cores	Sample rates will be approximately 10 Hz higher	Simplified fixation isolation
Define the AOIs as sections within the code rather than individual words	Gaze match will be more reliable if AOI is restricted to 105px or more	Data validity
Calibrate often and/or establish a reference point for standardization	The risk of spatially shifted datasets will decrease	Data validity and easier identification/correction of shifted data
Give comprehensive and easy-to-follow instructions	Participants who are aware of the negative environmental variables can take action to reduce them	Data accuracy
Encourage participants to rest their chin in their hand	The hand will work as a makeshift headrest and help stabilization as well as prevent fatigue	Decrease occurrences of shifted datasets

mostly be attributed to the lack of support for head movements and a suitable tested fixation algorithm. Accuracy was not outside a usable range for some software eye-tracking studies, but data quality was not high enough to match gaze to single rows or words. The collected data was, though not inconclusive, and allowed us to learn some very valuable lessons. Table 10 lists our concluding recommendations for researchers considering using the methodology to obtain the best results.

Studying code stimuli could be described as some of the slightest variations studied in eye-tracking. Therefore, these stimuli must be measured by the highest-standard equipment. Even when working with large AOIs, such as chunks of code, the metrics that describe comprehension and attention cannot correctly be applied because of data validity questions transferred from previous processing steps.

Even though the webcam eye-tracking was proven insufficient, the experiment also showed good potential for using webcams and client-side data collection to protect the participant's privacy. Once suitable solutions to the outlined problems are implemented, much of the doubt regarding validity will be removed. As both hardware and software are constantly evolving, it is not unreasonable to expect a future increase in the quality of collected data. A baseline of sufficient quality needs to be established to guarantee the integrity and reliability of the recorded data, preferably by validating its results with proven eye-tracking methods. As of today, webcam eye-tracking and eye-camera-tracking quality vary considerably, and given the scientific evidence so far, we cannot recommend the conduct of uncontrolled remote eye-tracking studies, even if applying limitations on the equipment and environment of the participant.

CRediT authorship contribution statement

Eva Thilderkvist: Writing – review & editing, Writing – original draft, Visualization, Validation, Software, Resources, Project administration, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Felix Dobslaw:** Writing – review & editing, Writing – original draft, Validation, Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

We have shared the link to both data and algorithms.

References

[1] U. Obaidellah, M. Al Haek, P. Cheng, A survey on the usage of eye-tracking in computer programming, ACM Comput. Surv. 51 (2018) 1–58, http://dx.doi.org/10.1145/3145904.

[2] K.R. Chandrika, J. Amudha, S.D. Sudarsan, Recognizing eye tracking traits for source code review, in: 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation, 2017, pp. 1–8, http://dx.doi.org/10.1109/ETFA.2017.8247637.

[3] S. D'Angelo, A. Begel, Improving communication between pair programmers using shared gaze awareness, in: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, CHI '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 6245–6290, http://dx.doi.org/10.1145/3025453.3025573.

[4] S. Papavlasopoulou, K. Sharma, M. Giannakos, L. Jaccheri, Using eye-tracking to unveil differences between kids and teens in coding activities, in: Proceedings of the 2017 Conference on Interaction Design and Children, IDC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 171–181, http://dx.doi.org/10.1145/3078072.3079740.

[5] N. Peitek, J. Siegmund, S. Apel, What drives the reading order of programmers?: An eye tracking study, in: ICPC '20: 28th International Conference on Program Comprehension, 2020, pp. 342–353, http://dx.doi.org/10.1145/3387904.3389279.

[6] K. Pernice, J. Nielsen, How to Conduct Eyetracking Studies, Tech. Rep., Nielsen Norman Group, Fremont CA 945397498 USA, 2009.

[7] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, A systematic literature review on the usage of eye-tracking in software engineering, Inf. Softw. Technol. 67 (2015) 79–107.

[8] T. Fritz, A. Begel, S.C. Müller, S. Yigit-Elliott, M. Züger, Using psycho-physiological measures to assess task difficulty in software development, in: Proceedings of the 36th International Conference on Software Engineering, ICSE 2014, Association for Computing Machinery, New York, NY, USA, 2014, pp. 402–413, http://dx.doi.org/10.1145/2568225.2568266.

[9] H. Uwano, M. Nakamura, A. Monden, K.I. Matsumoto, Analyzing Individual Performance of Source Code Review using Reviewers' Eye Movement: ETRA 2006 - Symposium on Eye Tracking Research and Applications, Proceedings - ETRA 2006, 2005, pp. 133–140.

[10] K. Semmelmann, S. Weigelt, Online webcam-based eye tracking in cognitive science: A first look, Behav. Res. Methods 50 (2) (2018) 451–465, http://dx.doi.org/10.3758/s13428-017-0913-7.

[11] P. Xu, K.A. Ehinger, Y. Zhang, A. Finkelstein, S.R. Kulkarni, J. Xiao, TurkerGaze: Crowdsourcing saliency with webcam based eye tracking, 2015, ArXiv.

[12] A. Papoutsaki, P. Sangkloy, J. Laskey, N. Daskalova, J. Huang, J. Hays, WebGazer: Scalable Webcam Eye Tracking Using User Interactions, in: 25th International Joint Conference on Artificial Intelligence, IJCAI 2016, New York City, New York, 2016.

[13] M. Ahrens, K. Schneider, M. Busch, Attention in software maintenance: An eye tracking study, in: 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming, EMIP, 2019, pp. 2–9, http://dx.doi.org/10.1109/EMIP.2019.00009.

[14] A. Jbara, D.G. Feitelson, How programmers read regular code: A controlled experiment using eye tracking, in: 2015 IEEE 23rd International Conference on Program Comprehension, 2015, pp. 244–254, http://dx.doi.org/10.1109/ICPC.2015.35.

[15] X. Luan, B. Zhang, D. Liu, X. Liu, X. Tong, K. Li, A lightweight heatmap-based eye tracking system, in: 2021 International Conference on Computer Communications and Networks, ICCCN, 2021, pp. 1–9, http://dx.doi.org/10.1109/ICCCN52240.2021.9522300.

[16] M. Ahrens, K. Schneider, Improving requirements specification use by transferring attention with eye tracking data, Inf. Softw. Technol. 131 (2021) 106483.

[17] R. Petrusel, J. Mendling, H.A. Reijers, Task-specific visual cues for improving process model understanding, Inf. Softw. Technol. 79 (2016) 63–78.

[18] M.A. Just, P.A. Carpenter, A theory of reading: From eye fixations to comprehension, Psychol. Rev. 87 (4) (1980) 329–354, http://dx.doi.org/10.1037/0033-295X.87.4.329.

[19] R. Andersson, L. Larsson, K. Holmqvist, M. Stridh, M. Nyström, One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms, Behav. Res. Methods 49 (2) (2017) 616–637, http://dx.doi.org/10.3758/s13428-016-0738-9.

[20] R.S. Hessels, D.C. Niehorster, C. Kemner, I.T.C. Hooge, Noise-robust fixation detection in eye movement data: Identification by two-means clustering (I2MC), Behav. Res. Methods 49 (5) (2017) 1802–1823, http://dx.doi.org/10.3758/s13428-016-0822-1.

[21] E. Thilderkvist, F. Dobslaw, On current limitations of online eye-tracking to study the visual processing of source code, 2022, Available at SSRN 4051688.

[22] T. Ribeiro, S. Brandl, A. Søgaard, N. Hollenstein, Webqamgaze: A multilingual webcam eye-tracking-while-reading dataset, 2023, arXiv preprint arXiv:2303.17876.

[23] M. Vos, S. Minor, G.C. Ramchand, Comparing infrared and webcam eye tracking in the visual world paradigm, 2022.

[24] M.E. Crosby, J. Stelovsky, How do we read algorithms? A case study, Computer 23 (1) (1990) 25–35, http://dx.doi.org/10.1109/2.48797.

[25] J.-m. Burkhardt, F. Détienne, S. Wiedenbeck, Object-oriented program comprehension: Effect of expertise, Task Phase, Empir. Softw. Eng. (2002) 115–156.

[26] A.J. Ko, B. Uttl, Individual differences in program comprehension strategies in unfamiliar programming systems, in: 11th IEEE International Workshop on Program Comprehension, 2003, 2003, http://dx.doi.org/10.1109/WPC.2003.1199201.

[27] B. Sharif, G. Jetty, J. Aponte, E. Parra, An empirical study assessing the effect of seeit 3D on comprehension, in: 2013 First IEEE Working Conference on Software Visualization, VISSOFT, pp. 1–10, http://dx.doi.org/10.1109/VISSOFT.2013.6650519.

[28] K. Holmqvist, M. Nyström, R. Andersson, R. Dewhurst, H. Jarodzka, J. van de Weijer, Eye Tracking: A Comprehensive Guide to Methods and Measures, OUP Oxford, 2011.

[29] N. Peitek, J. Siegmund, C. Parnin, S. Apel, J.C. Hofmeister, A. Brechmann, Simultaneous measurement of program comprehension with fMRI and eye tracking: A case study, in: Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–10, http://dx.doi.org/10.1145/3239235.3240495.

[30] Z. Sharafi, B. Sharif, Y.-G. Guéhéneuc, A. Begel, R. Bednarik, M. Crosby, A practical guide on conducting eye tracking studies in software engineering, Empir. Softw. Eng. 25 (5) (2020) 3128–3174, http://dx.doi.org/10.1007/s10664-020-09829-4.

[31] K. Holmqvist, M. Nyström, F. Mulvey, Eye tracker data quality: What it is and how to measure it, in: Eye Tracking Research and Applications Symposium, ETRA, 2012, http://dx.doi.org/10.1145/2168556.2168563.

[32] D.C. Niehorster, R. Zemblys, T. Beelders, K. Holmqvist, Characterizing gaze position signals and synthesizing noise during fixations in eye-tracking data, Behav. Res. Methods 52 (6) (2020) 2515–2534, http://dx.doi.org/10.3758/s13428-020-01400-9.

[33] R. Andersson, M. Nyström, K. Holmqvist, Sampling frequency and eye-tracking measures: How speed affects durations, latencies, and more, J. Eye Mov. Res. (3) (2010) 1–12, http://dx.doi.org/10.16910/JEMR.3.3.6.

[34] J. Gómez-Poveda, E. Gaudioso, Evaluation of temporal stability of eye tracking algorithms using webcams, Expert Syst. Appl.: Int. J. 64 (C) (2016) 69–83, http://dx.doi.org/10.1016/j.eswa.2016.07.029.

[35] K. Harezlak, P. Kasprowski, Understanding eye movement signal characteristics based on their dynamical and fractal features, Sensors 19 (3) (2019) 626, http://dx.doi.org/10.3390/s19030626.

[36] K. Holmqvist, P. Blignaut, Small eye movements cannot be reliably measured by video-based P-CR eye-trackers, Behav. Res. Methods 52 (5) (2020) 2098–2121, http://dx.doi.org/10.3758/s13428-020-01363-x.

[37] B. Sharif, M. Falcone, J. Maletic, An eye-tracking study on the role of scan time in finding source code defects. http://dx.doi.org/10.1145/2168556.2168642.

[38] Z. Sharafi, T. Shaffer, B. Sharif, Y.-G. Guéhéneuc, Eye-tracking metrics in software engineering, in: 2015 Asia-Pacific Software Engineering Conference, APSEC, 2015, pp. 96–103, http://dx.doi.org/10.1109/APSEC.2015.53.

[39] A. Agarwal, D. JeevithaShree, K.S. Saluja, A. Sahay, P. Mounika, A. Sahu, R. Bhaumik, V.K. Rajendran, P. Biswas, Comparing two webcam-based eye gaze trackers for users with severe speech and motor impairment, in: A. Chakrabarti (Ed.), Research Into Design for a Connected World, Smart Innovation, Systems and Technologies, Springer, Singapore, 2019, pp. 641–652, http://dx.doi.org/10.1007/978-981-13-5977-4_54.

[40] N.T. Bott, A. Lange, D. Rentz, E. Buffalo, P. Clopton, S. Zola, Web camera based eye tracking to assess visual memory on a visual paired comparison task, Front. Neurosci. 11 (2017) 370, http://dx.doi.org/10.3389/fnins.2017.00370.

[41] N. Dubey, S. Setia, A.A. Verma, S. Iyengar, Wikigaze: Gaze-based personalized summarization of Wikipedia reading session, in: Proceedings of the 3rd Workshop on Human Factors in Hypertext, 2020, pp. 1–9, http://dx.doi.org/10.1145/3406853.3432662.

[42] A. Diaz-Tula, C.H. Morimoto, Robust, real-time eye movement classification for gaze interaction using finite state machines, in: 2017 COGAIN Symposium, 2017.

[43] M.E. Crosby, J. Scholtz, S. Wiedenbeck, The roles beacons play in comprehension for novice and expert programmers, in: Programmers, 14th Workshop of the Psychology of Programming Interest Group, Brunel University, 2002, pp. 18–21.

[44] J. Melo, F.B. Narcizo, D.W. Hansen, C. Brabrand, A. Wasowski, Variability through the eyes of the programmer, in: 2017 IEEE/ACM 25th International Conference on Program Comprehension, ICPC, 2017, pp. 34–44, http://dx.doi.org/10.1109/ICPC.2017.34.

[45] Z. Soh, F. Khomh, Y.-G. Guéhéneuc, G. Antoniol, B. Adams, On the effect of program exploration on maintenance tasks, in: 2013 20th Working Conference on Reverse Engineering, WCRE, 2013, pp. 391–400, http://dx.doi.org/10.1109/WCRE.2013.6671314.

[46] R. Turner, M. Falcone, B. Sharif, A. Lazar, An eye-tracking study assessing the comprehension of C++ and Python source code, in: Proceedings of the Symposium on Eye Tracking Research and Applications, ETRA '14, Association for Computing Machinery, New York, NY, USA, 2014, pp. 231–234, http://dx.doi.org/10.1145/2578153.2578218.

[47] Z. Sharafi, Z. Soh, Y.-G. Guéhéneuc, G. Antoniol, Women and men — Different but equal: On the impact of identifier style on source code reading, in: 2012 20th IEEE International Conference on Program Comprehension, ICPC, 2012, pp. 27–36, http://dx.doi.org/10.1109/ICPC.2012.6240505.

[48] R. Bednarik, Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations, Int. J. Hum.-Comput. Stud. 70 (2) (2012) 143–155, http://dx.doi.org/10.1016/j.ijhcs.2011.09.003.

[49] D. Binkley, M. Davis, D. Lawrie, J.I. Maletic, C. Morrell, B. Sharif, The impact of identifier style on effort and comprehension, Empir. Softw. Eng. 18 (2) (2013) 219–276, http://dx.doi.org/10.1007/s10664-012-9201-4.

[50] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J.H. Paterson, C. Schulte, B. Sharif, S. Tamm, Eye movements in code reading: Relaxing the linear order, in: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, ICPC '15, IEEE Press, Florence, Italy, 2015, pp. 255–265.

[51] P. Peachock, N. Iovino, B. Sharif, Investigating eye movements in natural language and C++ source code - A replication experiment, in: D.D. Schmorrow, C.M. Fidopiastis (Eds.), Augmented Cognition. Neurocognition and Machine Learning, in: Lecture Notes in Computer Science, Springer International Publishing, Cham, 2017, pp. 206–218, http://dx.doi.org/10.1007/978-3-319-58628-1_17.

[52] P. Romero, R. Cox, B. du Boulay, R. Lutz, 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, in: Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2002, pp. 221–235, http://dx.doi.org/10.1007/3-540-46037-3_23.

[53] J. Mucke, M. Schwarzkopf, J. Siegmund, REyeker: Remote eye tracker, in: ACM Symposium on Eye Tracking Research and Applications, ETRA '21 Short Papers, Association for Computing Machinery, New York, NY, USA, 2021, pp. 1–5, http://dx.doi.org/10.1145/3448018.3457423.

[54] R. Bednarik, M. Tukiainen, Validating the restricted focus viewer: A study using eye-movement tracking, Behav. Res. Methods 39 (2) (2007) 274–282, http://dx.doi.org/10.3758/BF03193158.

[55] S. Kokolakis, Privacy attitudes and privacy behaviour: A review of current research on the privacy paradox phenomenon, Comput. Secur. 64 (2017) 122–134, http://dx.doi.org/10.1016/j.cose.2015.07.002.

[56] J. Skansholm, Java Direkt Med Swing, sixth ed., Studentlitteratur, 2010.

[57] H. Schildt, Java-the Complete Reference, eleventh ed., McGraw-Hill Education, 2019.

[58] T. Nield, Learning RxJava, Packt Publishing Ltd., 2017.

[59] Various, The Project Gutenberg EBook of Grimm's Fairy Tales, The Project Gutenberg, 2016.

[60] M. Nyström, R. Andersson, K. Holmqvist, J. van de Weijer, The influence of calibration method and eye physiology on eyetracking data quality, Behav. Res. Methods 45 (1) (2013) 272–288, http://dx.doi.org/10.3758/s13428-012-0247-4.

[61] C. Chu, M. Rosenfield, J.K. Portello, J.A. Benzoni, J.D. Collier, A comparison of symptoms after viewing text on a computer screen and hardcopy, Ophthalmic Physiol. Opt. 31 (1) (2011) 29–32, http://dx.doi.org/10.1111/j.1475-1313.2010.00802.x.

[62] Saccadic eye movements, https://www.brown.edu/Departments/Engineering/Courses/122JDD/Lcturs/sacc05.html.

[63] J. Zagermann, U. Pfeil, H. Reiterer, Studying eye movements as a basis for measuring cognitive load, in: Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems, CHI EA '18, Association for Computing Machinery, New York, NY, USA, 2018, pp. 1–6, http://dx.doi.org/10.1145/3170427.3188628.

[64] D.D. Salvucci, J.H. Goldberg, Identifying fixations and saccades in eye-tracking protocols, in: Proceedings of the 2000 Symposium on Eye Tracking Research & Applications, ETRA '00, Association for Computing Machinery, New York, NY, USA, 2000, pp. 71–78, http://dx.doi.org/10.1145/355017.355028.

[65] K. Ooms, V. Krassanakis, Measuring the spatial noise of a low-cost eye tracker to enhance fixation detection, J. Imaging 4 (8) (2018) 96, http://dx.doi.org/10.3390/jimaging4080096.

[66] N. Galley, D. Betz, C. Biniossek, Fixation durations - Why are they so highly variable?, 2015, pp. 83–106.

[67] T. Busjahn, C. Schulte, A. Busjahn, Analysis of code reading to gain more insight in program comprehension, in: Proceedings of the 11th Koli Calling International Conference on Computing Education Research, Koli Calling '11, Association for Computing Machinery, New York, NY, USA, 2011, pp. 1–9, http://dx.doi.org/10.1145/2094131.2094133.