



HAL
open science

Visually Analyzing Eye Movements on Natural Language Texts and Source Code Snippets

Tanja Blascheck, Bonita Sharif

► **To cite this version:**

Tanja Blascheck, Bonita Sharif. Visually Analyzing Eye Movements on Natural Language Texts and Source Code Snippets. ETRA 2019 - ACM Symposium on Eye Tracking Research & Applications, Jun 2019, Denver, United States. 10.1145/3314111.3319917 . hal-02084109

HAL Id: hal-02084109

<https://inria.hal.science/hal-02084109v1>

Submitted on 3 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Visually Analyzing Eye Movements on Natural Language Texts and Source Code Snippets

Tanja Blascheck
Inria
Saclay, France
tanja.blascheck@inria.fr

Bonita Sharif
University of Nebraska-Lincoln
Lincoln, Nebraska 68588, USA
bsharif@unl.edu

ABSTRACT

In this paper, we analyze eye movement data of 26 participants using a quantitative and qualitative approach to investigate how people read natural language text in comparison to source code. In particular, we use the radial transition graph visualization to explore strategies of participants during these reading tasks and extract common patterns amongst participants. We illustrate via examples how visualization can play a role at uncovering behavior of people while reading natural language text versus source code. Our results show that the linear reading order of natural text is only partially applicable to source code reading. We found patterns representing a linear order and also patterns that represent reading of the source code in execution order. Participants also focus more on those areas that are important to comprehend core functionality and we found that they skip unimportant constructs such as brackets.

CCS CONCEPTS

• **Human-centered computing** → **Visual analytics**; *Interactive systems and tools*; *Visualization theory, concepts and paradigms*.

KEYWORDS

Eye tracking, program comprehension, visual analysis, visualization

ACM Reference Format:

Tanja Blascheck and Bonita Sharif. 2019. Visually Analyzing Eye Movements on Natural Language Texts and Source Code Snippets. In *2019 Symposium on Eye Tracking Research and Applications (ETRA '19)*, June 25–28, 2019, Denver, CO, USA. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3314111.3319917>

1 INTRODUCTION

Software developers work with systems that consist of a variety of artifacts such as source code, requirements, and design documents. The source code plays an important role in software because it is the main artifact that represents the true functionality of a system. Program comprehension [Letovsky and Soloway 1986] is an activity that developers engage in while they read the source code with the goal of trying to understand what it does. Often times, code is not necessarily modified by the original developer who wrote it. This means the developer needs to become familiar with the code they did not necessarily write. The process during which developers read and navigate the code to make sense of what the code is doing helps them build a better mental model of the code. This activity is crucial for developers to become familiar with a software system before they make a change.

Program comprehension and source code reading is a precursor to many software engineering activities such as bug fixing and code review. Even though words from a natural language such as English appear in source code, they do not appear in the same type of structure as they would in a natural language paragraph. Software engineering researchers have been using eye tracking to collect eye movement data of developers to investigate what they read and what they focus on while they perform various tasks.

To analyze differences in reading behavior between natural language text and source code, we present the results of an eye tracking study in this paper, during which participants had to read natural language texts and short code snippets on a computer monitor. This study is a replication, using different participants and a different programming language, of an earlier study by Busjahn et al. [2015]. The main research question we seek to answer is: *How can we best analyze eye movement data for short texts such as natural language and source code snippets to find common patterns?* In contrast to the Busjahn et al. study, in which they solely performed an analysis with linearity metrics to investigate differences between reading behavior, we analyze the eye movement data we collected using both a quantitative and a qualitative approach. A qualitative analysis using a state-of-the-art visualization technique namely the *radial transition graph* helps us to extract common patterns that are hard to detect with quantitative analysis alone. In addition, initial results and insights found through qualitative analysis can be confirmed with quantitative results. Therefore, we use a combination of both analyses to explore participants' strategies, find common patterns, and analyze the behavior of novices and non-novices reading natural language texts versus source code snippets. Our results show that visualizations can play a significant role to help us understand large eye tracking datasets for these types of studies.

2 RELATED WORK

We discuss related work from the program comprehension literature that studies source code reading using eye tracking, and visualization techniques for analyzing eye movement data.

2.1 Eye Tracking in Program Comprehension

Many studies have been conducted within the software engineering community to understand program comprehension. We refer the reader to a systematic literature review on studies done in programming and software engineering using eye tracking [Obaidallah et al. 2018; Sharafi et al. 2015]. Here, we focus only on papers that investigate source code. Overall, researchers have studied many different programming languages (C [Sharif et al. 2012; Uwano et al. 2006, 2007], C++ [Turner et al. 2014], Java [Aschwanden and Crosby 2006; Bednarik 2012; Busjahn et al. 2015, 2014a, 2011, 2014b; Sharafi et al.

2012], Pascal [Crosby and Stelovsky 1990], Python [Turner et al. 2014], etc.), with Java being the most investigated language. Some typical tasks that previous work has considered to better understand program comprehension are detection of bugs [Sharif et al. 2012; Turner et al. 2014; Uwano et al. 2006, 2007], code summaries (i.e., describing what a specific source code is doing) [Aschwandt and Crosby 2006; Busjahn et al. 2015; Turner et al. 2014], or code comprehension by asking participants to answer a comprehension question [Busjahn et al. 2015, 2014a, 2011, 2014b] or a fill-in-the-blanks test [Crosby et al. 2002; Crosby and Stelovsky 1990].

In addition, understanding the differences between novice and expert programmers [Busjahn et al. 2015, 2014a, 2011, 2014b; Crosby et al. 2002; Crosby and Stelovsky 1990; Sharif et al. 2012; Turner et al. 2014] helps researchers in program education. For example, studying differences of novices and experts has provided insights into expert behavior who focus more on relevant parts of source code (referred to as *beacons* [Brooks 1983]) than novices [Aschwandt and Crosby 2006; Crosby et al. 2002].

Similar to this notion of beacons, previous work has also focused on the difference of comments and other code structures (i.e., keywords, operators, identifiers, numbers). Comments help with program comprehension, however, Crosby and Stelovsky [1990] found that there are different groups of participants: code- and comment-oriented participants, for both novices and experts. Busjahn et al. [2011] found that operators and identifiers received more attention than keywords and numbers. In addition, Busjahn and colleagues [Busjahn et al. 2015, 2014a, 2011, 2014b] have run multiple experiments to analyze the difference in reading behavior of source code in comparison to natural language text. In their most recent work, Busjahn et al. [2015] introduce several linearity metrics to determine if people read source code the same way as they read natural language text. They found that experts read source code in a less linear fashion compared to novices and novices read source code less linearly than natural language text.

The study presented in this paper replicates the study by Busjahn et al. [2015]. The main difference between their study and our study is that we use C++ instead of Java for the code snippets. The code snippets were converted to C++ without modifying the main intent of the code. Another difference is the studied population. All of our novices were first year computer science students. Busjahn et al. recruited novices from a beginner course in Java for non Computer Science students. We explore general strategies of participants and extract patterns from the eye movement data, by not only focusing on metrics but also applying a visualization technique to show linear vs. non-linear reading.

2.2 Visualizing Eye Tracking Data

Analyzing eye movement data collected in a study can benefit from visual approaches to explore participants' strategies and behavior. Blascheck et al. [2017a] collected an exhaustive list of visualization techniques for eye movement data dividing these approaches into point-based (focusing on fixations and saccades) and Area of Interest (AOI) based methods. We use AOIs to visually analyze eye movement data and, therefore, only focus on related techniques using AOIs. In general, we are interested in the relation between AOIs, for example, which AOIs have the most transitions between each

other, which AOIs were focused on the longest, and the temporal order when viewing AOIs.

Visual approaches investigating reading behavior such as Alpcarf [Yang and Wacharamanatham 2018] use a hierarchical order of AOIs to show strategies of participants reading a scientific research paper. However, this approach does not support an exploration of participant strategies and our short texts would not allow to create a sufficient AOI hierarchy. Spakov et al. [2017] use another approach to analyze activity of people learning to read. They integrate five visualization techniques to analyze the eye movement data collected in a reading study—dynamic gaze and word replay, gaze plots, word reading duration, and summaries. However, they do not focus on the sequence of AOIs and do not explore the strategies of participants. Clark et al. present iTraceVis [Clark and Sharif 2017], a visualization technique that is built into the Eclipse IDE and visually represents eye movement data collected while participants are programming. This visualization technique is not restricted to short code snippets. It is built on top of iTrace [Guarnera et al. 2018; Shaffer et al. 2015] and supports four main views—a heatmap view, a gaze skyline, a static gaze map, and a dynamic gaze map.

In this paper, we take a different approach to visualize our data. Because our study dealt with short code snippets, we did not use the visualization techniques presented above. Instead, we use the *radial transition graph* [Blascheck et al. 2017b] to determine and inspect transition sequences. This technique is available online (<http://www.rtgct.fbeck.com/>) and allows us to upload our own eye movement data. The radial transition graph is a donut chart, depicting AOIs as segments. Each segment (i.e., AOI) has a unique color. The size of the donut segments either corresponds to the dwell time or it can display all segments with equal sizes. Transitions between AOIs are depicted as arcs. Transitions between AOIs, are separated into outgoing and incoming transitions using two anchors—black and white circles respectively. These separate anchors avoid the usage of arrow heads for the direction of a transition and reduces visual clutter. The thickness of the arcs represents the transition count (number of transitions between two AOIs).

Figure 3 shows an overview of radial transition graphs for a selection of participants for two stimuli as an example of the visualization technique. Selecting a graph in the list, shows it on the right side of the list in a detail view. This detail view shows an individual radial transition graph with a scarf plot [Holmqvist et al. 2011] below it, which represents the temporal order of AOIs (cf. Figure 4 right). An analyst can interact with the scarf plot and look at each transition individually (transition slider) or restrict the time range for the transitions (time slider).

3 EYE TRACKING STUDY

The goal of our eye tracking study is to understand how natural language text and source code reading differ. We are also interested in how novices and non-novices read source code, if there are differences in their reading behavior, and if we can extract some common patterns. Next, we present the study design, information about participants, the AOIs we defined for our analysis, and our data cleansing approach.

Table 1: Participants' programming background gathered from the background questionnaire.

How well do you understand programming?		How well do you understand C++?		How often do you program in C++ per week?	
Not at all	11	Not at all	6	Never	8
Very little	5	Very little	5	1-2 hours	5
Fairly well	3	Fairly well	12	3-4 hours	12
Very well	6	Very well	2	5+	0

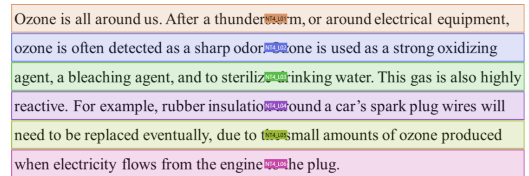
3.1 Study Design and Tasks

We designed our study similar to Busjahn et al. [2015] and asked participants to read natural language texts as well as source code snippets. Our stimuli were shown on a computer monitor allowing us to record eye movement data while participants read the presented text. We chose four natural language texts from the psychology literature with 6 to 12 lines per text. Also, participants had to read and comprehend four C++ source code snippets. We chose C++ as the programming language for the study, because this is taught as the first language at the university where we conducted the study, however, not all students take this course in their first semester. All source code snippets present concepts undergraduates learn in a beginners programming class. The first source code SC 2 (7 lines) is a simple program adding and subtracting numbers. Source code SC 5 (20 lines, cf. Figure 4 left) is more advanced and outputs a list of numbers in reverse order. SC 6 (25 lines) is a class definition of a vehicle with an accelerate-function, and lastly, SC 7 (11 lines, cf. Figure 5 left) replaces the word *World* by *Sun* in the text string *Hello World!* No comments were included to ensure that participants focus on the source code to understand it. Also, the source code was shown in black-and-white without syntax highlighting to avoid any confounding factors. All stimuli had the same size (1280 px × 720 px). Because the number of lines per stimulus varied between 6 and 25, we adjusted the font size and spacing to make the text clearly legible in the predefined area.

The task given to participants plays a major role in determining how they read text, therefore, we told participants that they would be asked a reading comprehension multiple choice question for the natural language text and a question about the source code's behavior including output produced in multiple choice form. We told participants to study the stimuli for as long as they wanted. Once they were done reading the text they had to answer the comprehension question in an online questionnaire.

3.2 Participants

We cannot use the general population as our participant pool because one of the requirements of the study was that our participants are able to read and understand source code. Therefore, we recruited mostly students from a local university. Overall, 26 participants performed the study (6 females, 20 male). The age range was equally distributed, with 11 participants between 20-22, 5 participants between 23-25, and 9 older than 25. All participants, except two who had a medium understanding of English, indicated that they had a high understanding of English and 21 indicated that English was


Figure 1: AOIs (shown as colored rectangles on top of the stimulus) for natural language text NT 4.

their native language. Participants were not compensated monetarily for their participation in the study, however, students were offered extra credit by their instructor. None of the authors of this paper served as the grade authority for the students. The average time to complete the study was approximately 20 minutes.

As is common in program comprehension studies [Busjahn et al. 2015, 2011; Crosby et al. 2002; Crosby and Stelovsky 1990; Sharif et al. 2012; Turner et al. 2014], we want to analyze differences between novice and non-novice programmers. To distinguish these two groups of programmers, we asked participants to rate their experience with programming. Table 1 summarizes participants' programming background. For our analysis, we define non-novices as participants who have taken 5+ programming courses. Based on participants' self-reporting of their experience with programming and our definition of non-novices, our study consists of ten non-novices (P02, P03, P05, P07, P08, P10, P13, P23, P25, P26).

3.3 Technical Setup

A Tobii X60 eye tracker recorded eye movement data with a sampling rate of 60 Hz on a screen with a resolution of 1920 px × 1080 px. We used Tobii Studio to gather data for the study. Tobii Studio automatically aggregates the raw eye movement data into fixations using the I-VT filter (velocity threshold = 30 degrees / second). To calibrate the eye tracker, we conducted a nine-point calibration with each participant before they began the task. We exported all raw data from Tobii Studio as tab-separated value files. The stimuli, AOIs, and replication package for the study are available at <http://seresl.unl.edu/ETRA2019> along with additional visualizations for each stimulus.

3.4 Study Procedure

We completed an Institutional Review Board (IRB) protocol for the study and ran it for two semesters at a local university with undergraduate students as well as some invited external programmers. First, participants had to fill out a questionnaire about their background information. Next, we performed the calibration for each participant before they started with the tasks. Each participant saw the natural language texts and source code snippets in random order. Between each stimulus, participants had to answer a question presented in an online questionnaire.

3.5 Data Preparation

Before we analyzed the eye movement data, we ran the data through a general data cleansing step. Due to different reasons, for three recordings, the percentage of gaze samples was below 80%. We, therefore, removed these three participants from further analyses (P21, P24, P26). Also, investigating the recordings of all participants,

the data of two participants (P11, P22) for the source code stimuli was too low. These participants had completion times of ≤ 6 s, had given at least three incorrect answers, and had looked at $\leq 50\%$ of the AOIs on average. We excluded them from the analysis as well. This leaves us with a total of 21 participants (5 female, 16 male), of which, nine participants are non-novices (2 female, 7 male).

In addition to removing five participants completely from the analysis, for some participants we had to correct the eye movement data due to drift [Palmer and Sharif 2016]. One author visually inspected all scanpaths of participants. If the scanpath had an obvious offset from the lines of text, usually present by fixations being below the last line of the stimulus, she adjusted the scanpaths accordingly to match the lines of the stimulus, else the assignment of fixations to AOIs would have been incorrect. This was done carefully by visually inspecting a set of scanpaths. Because of the presence of text lines this procedure was fairly straight-forward. Overall, we had to adjust the scanpaths of 15 participants and we shifted them on average about 26 px. Note, that we did not cherry pick individual fixations to move but rather moved complete scanpaths that were obviously off. This kind of drift happens mainly vertically causing the AOI to match incorrectly had they not been corrected. A similar approach was also taken by Busjahn et al. [2015].

Next, we assigned fixations to AOIs. We defined each line of the natural language text and each line of source code as an individual AOI. Figure 1 depicts an example of a natural language text stimulus with AOIs shown as colored rectangles and Figure 4 presents the AOIs as colored rectangles for a source code stimulus.

4 EXPERIMENTAL RESULTS

We first start with a general description of the accuracy and completion times of participants. Next, we inspect the eye movement data in more detail and investigate the AOI coverage. Lastly, we investigate the reading behavior participants exhibited when reading natural language text versus source code. We are especially interested in finding common patterns for the source code and inspect if there are differences between novices and non-novices.

We analyze and report some metrics using interval estimation [Dragicevic 2016] with sample means and 95% confidence intervals (CIs). We can be 95% confident that this interval includes the population mean. These results are highly representative of the plausible values of the true population mean and the approach supports future replication efforts. We use BCa bootstrapping to construct confidence intervals (10,000 bootstrap iterations). We analyze the CIs using estimation techniques, i. e., interpreting them by providing different strengths of evidence about the population mean, as recommended in the literature [Besançon and Dragicevic 2017; Cumming 2013; Dragicevic 2016; Gigerenzer 2004; Goodman 1999; Schmidt and Hunter 1997]. Nonetheless, a p-value approach of our technique can be obtained following the recommendations from Krzywinski and Altman [2013].

4.1 Accuracy Rate and Completion Times

On average the accuracy (max = 4 correct responses) for the natural language texts was 3.3 and for the source code snippets 2.86. Figure 2A (top) displays weak evidence for a better accuracy of natural language texts. Comparing the accuracy of the source code

snippets based on expertise, the average for novices is 2.5 and 3.3 for non-novices. Figure 2A (bottom) displays weak evidence for a higher accuracy of non-novices of source code.

The average completion time for the natural language texts and source code snippets was 27.4 s and 32.5 s respectively. Figure 2B (top) shows weak evidence for a shorter completion time of natural language texts. The average completion times of novices was 25.7 s and for non-novices 41.6 s. Figure 2B (bottom) shows strong evidence for a shorter completion time of novices for the source code stimuli.

4.2 Line (AOI) Coverage

An initial inspection of the radial transition graphs revealed that some participants did not focus on some of the AOIs in the stimuli, for both the natural language texts and source code snippets. Therefore, we calculate the AOI coverage, which gives us the percentage of AOIs that have not been focused on. This is similar to Busjahn et al. [2015], who calculate the element coverage, which is the percentage of words that were focused on. For the natural language texts we do not distinguish between novices and non-novices and the average AOI coverage is 98.3%.

For the source code stimuli the average AOI coverage for novices is 89.2% and for non-novices is 88.5%. The AOIs, which some participants did not focus on are the include statement(s) or namespace statement (9 novices / 8 non-novices at least once for the four source code stimuli), the closing brackets (11 novices / 9 non-novices), the main-function header (5 novices / 2 non-novices), and sometimes also the return-statements (6 novices / 4 non-novices). In the following, we refer to these specific statements as code constructs.

Figure 2C shows strong evidence for higher AOI coverage of the natural language texts (top) and weak evidence for a higher AOI coverage of the source code AOIs for novices (bottom).

Similar to previous work [Aschwanden and Crosby 2006; Busjahn et al. 2011; Crosby and Stelovsky 1990], we look at specific constructs of the source code. However, because we look at the source code on a line-by-line basis instead of individual words, we do not have a fine grained analysis of keywords, identifiers, comments etc. Because there is a different amount of lines for each of the code constructs, we average the total dwell time spent on each for a comparison. The average dwell time on the header is 11.6% for novices and 7.2% for non-novices; for closing brackets it is 7.0% for novices and 3.5% for non-novices; the main-function receives on average 25.4% from novices and 25.8% from non-novices; the return statement receives 12.8% from novices and 7.7% from non-novices; and the average total dwell time for the actual source code for novices is 43.2% and for non-novices it is 55.8%.

4.3 Natural Language: Linear Reading Order

The natural reading order when reading a text in Latin languages is from left to right and top to bottom. Although, source code follows the same writing direction, source code is not necessarily read in such a distinct linear reading fashion. In addition, previous work on source code reading [Busjahn et al. 2015] found that experts deviate from this linear reading path more than novices. Therefore, we analyze this difference for natural language text and source code

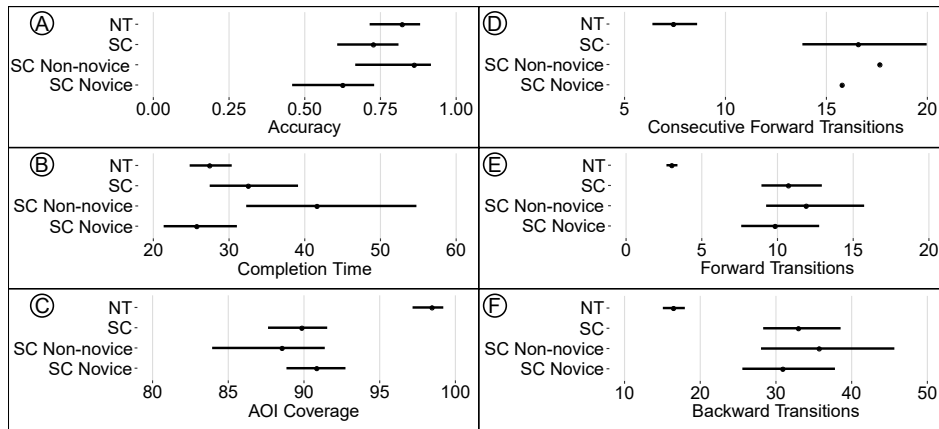


Figure 2: A) Accuracy, B) completion time, C) AOI coverage, D) consecutive and E) forward transitions, and F) backward transitions for NT and SC as well as non-novices and novices. Error bars: 95% Bootstrap confidence intervals (CIs).

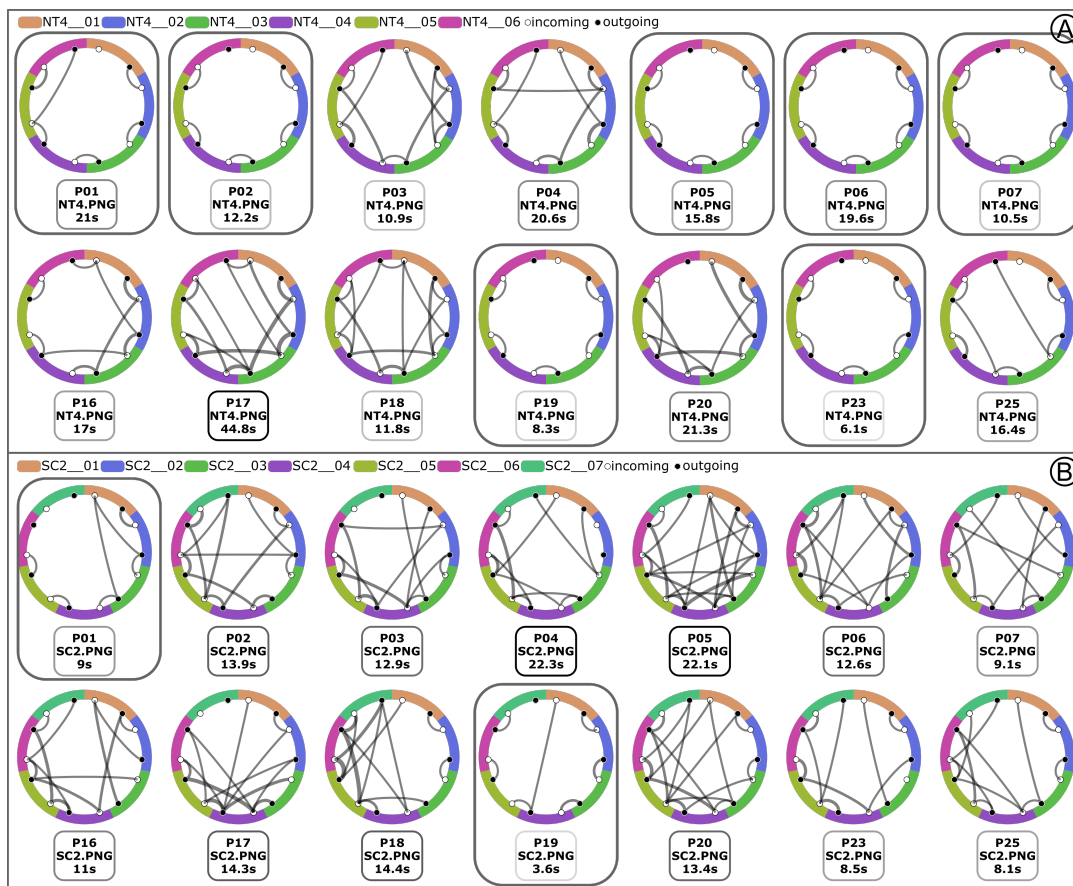


Figure 3: Overview of the radial transition graphs for a selection of participants for stimulus NT 4 (A) and SC 2 (B). We highlight radial transition graphs with a gray boarder that show a linear reading, i.e., for each AOI there is an arc (transition) from the outgoing (black circle) to the incoming (white circle) of the consecutive AOI. For demonstration purposes, we manually edited the data of some participants (e.g., removed the beginning/ending) to emphasize the linear reading order. Typically, in the beginning, participants jump to the start of the text because they had to answer a question before looking at the next stimulus.

as well as inspect the contrasting type of reading of source code for novices and non-novices.

First, we calculate the average number of forward transitions between consecutive AOIs as well as forward and backward transitions to any other AOI. This gives us an estimate of linear reading of AOIs. For the natural language texts, for all participants the average number of forward transitions to consecutive AOIs is 7.4, 3.0 for forward transitions to any other AOI, and for backward transitions 16.5. The top part of Figure 2D-F shows strong evidence of less transitions (forward consecutive, forward, and backward) for the natural language texts.

Using the radial transition graph, we further analyze the linear reading order, by looking at arcs from two consecutive AOIs (from the black to the white circle) as highlighted with a gray border in Figure 3A. Because participants input the answer from the previous question before moving on to the next stimulus, the first fixations on the new stimulus are usually used to jump to the beginning of the text. Therefore, we exclude them from the analysis by constraining the time slider. For the natural language text, searching for this patterns, we can find such a linear reading for most participants (NT 2: 19; NT 3: 18; NT 4: 18; NT 10: 17). This complements the analysis from above, however, by inspecting the scarf plots shown below each radial transition graph in the details view, we also find that typically this linear reading was mainly done in the beginning of a stimulus inspection. After participants have read the text once from beginning to end, some participants re-read or skimmed the text a second time. This skimming does not always follow a linear fashion as the initial first reading. Therefore, some radial transition graphs shown in Figure 3A also have crossing transitions indicating that participants jumped between non-consecutive AOIs.

4.4 Analysis of Source Code

Inspecting the source code data using the radial transition graphs (cf. Figure 3B for SC 2), we find less participants who read the code from top to bottom in a linear order. Therefore, we first summarize some findings on linear reading order and then focus more on common patterns and individual differences between participants' strategies. Due to space limitations we only discuss three of the four source code stimuli.

4.4.1 Linear Reading Order. We calculate the average number of forward transitions between consecutive AOIs as well as forward and backward transitions between other AOIs for both novices and non-novices for the source code stimuli. The average number of forward transitions between consecutive AOIs for novices is 15.77 and for non-novices is 17.6; the average number of forward transitions to any other AOI for novices is 9.8 and for non-novices is 11.9; and the average number of backward transitions to any other AOI for novices is 30.90 and for non-novices is 35.7. Figure 2D (bottom) shows strong evidence of less consecutive forward transitions for novices. Figure 2E and F (bottom) shows weak evidence of less forward and backward transitions for novices.

Again inspecting the individual radial transition graphs, we find only 2 participants for SC 2 (cf. Figure 3B), 5 for SC 5, and 2 for SC 7 reading the source code from top to bottom. This is different for SC 6: here we can find 10 participants which followed a linear first reading of the code. For SC 2, SC 5, and SC 7 most of the participants

depicting a linear reading direction were novices, again this was different for SC 6, for which more non-novices (5 of the 9 non-novices) linearly read this source code snippet as opposed to 5 of the 12 novices. Figure 4 (right) shows an example of participant P19 depicting this strategy for SC 5, which is shown by the arcs (from black to white circles) for consecutive AOIs.

4.4.2 SC 5: Output List of Numbers in Reverse Order. For this stimulus, we want to focus on participants' strategies and find common patterns participants applied to solve the task. We can find that almost all participants focused in linear order on the relevant AOIs for the three functions (doSomething, print, and main) of the code (cf. Figure 4 left). The following patterns reflect this:

- SC5_L04 → SC5_L05 → SC5_L06 → SC5_L07 (doSomething-method; 18 participants)
- SC5_L11 → SC5_L12 → SC5_L13 (print-method; 20 participants)
- SC5_L16 → SC5_L17 → SC5_L18 (main-method; 19 participants)

We can also find a longer pattern for seven participants that includes focusing on the complete main-method in linear order: SC5_L15 → SC5_L16 → SC5_L17 → SC5_L18 → SC5_L19.

Similar to Busjahn et al. [2015] these patterns can be classified as *story order*. We also want to extract patterns that depict the *execution order*, i.e., the order in which the code is executed. For SC 5 this means, for example, that participants, after focusing on SC5_L17, jump to the doSomething-method and focus on the AOIs for this method (SC5_L05 → SC5_L06 → SC5_L07). We find this pattern for both novices and non-novices (overall 10 participants). The same applies to the pattern SC5_L18 → SC5_L11 → SC5_L12 → SC5_L13 (14 participants, both novices and non-novices), which has participants jumping to the print-method after this method is being called in the main-method. In addition, 11 participants (both novices and non-novices) switch between SC5_L12 → SC5_L13 (for loop to print output) multiple times.

4.4.3 SC 6: Vehicle Class with an Accelerate Function. Because SC 6 consists of 25 AOIs, we create five higher level AOIs to analyze this stimulus. The AOIs contain the AOIs of the header (SC6_header), the class statement with attributes (SC6_class), the two functions (SC6_vehicle, SC6_accelerate), and the main-method (SC6_main).

Analyzing the data based on these higher level AOIs, we detect that eight participants paid most attention to SC6_class, nine participants on SC6_vehicle, and four participants to SC6_accelerate. Searching for common patterns, we find the pattern SC6_vehicle → SC6_accelerate → SC6_main in this or similar form for all participants except P23. Two other interesting patterns are the focus on SC6_class followed by a sequence of SC6_vehicle ↔ SC6_accelerate, which shows that participants were switching back and forth between the two functions to understand the code. Similarly, this was done as a switching between SC6_vehicle ↔ SC6_accelerate multiple times before focusing on SC6_main.

4.4.4 SC 7: Replace Text in a String. Eight participants answered the comprehension question to source code snippet SC 7 incorrectly. However, we have to distinguish participants based on their answer. The source code was a string replacement algorithm, in which the word *World* was replaced by the word *Sun*. Participants had to answer the question which word or character is replaced in

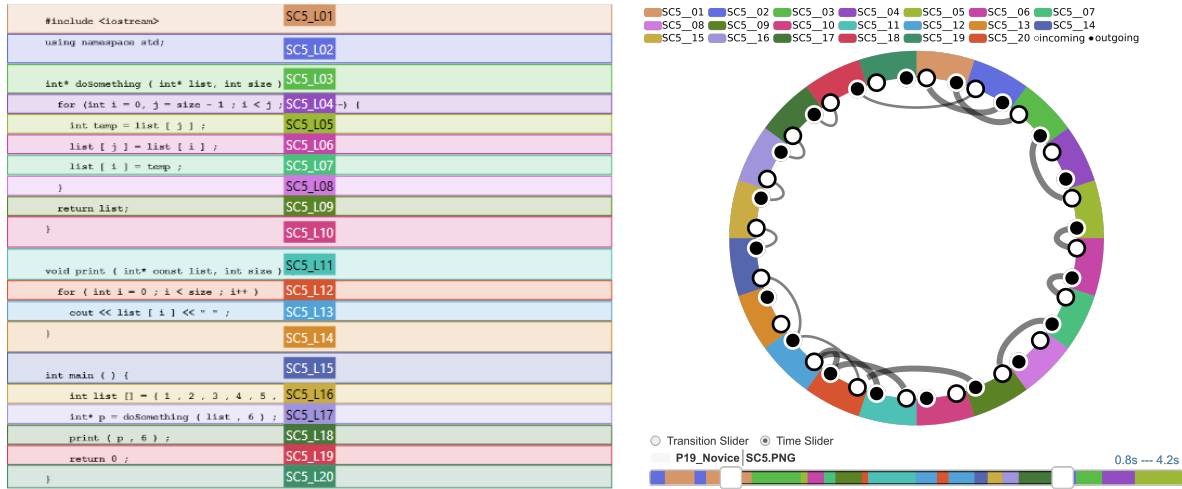


Figure 4: Stimulus SC 5 with AOIs shown as colored rectangles on top of the stimulus (left). More or less linear reading of SC 5 (arcs from the black to the white circle of consecutive AOIs) for participant P19 (right).

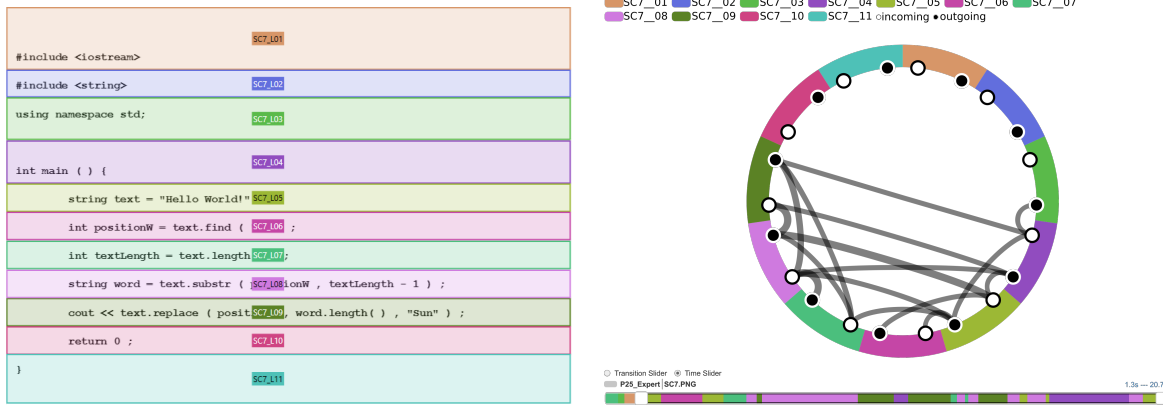


Figure 5: Stimulus SC 7 with AOIs shown as colored rectangles on top of the stimulus (left). A typical pattern participant P25 used for SC 7 is to focus on the relevant AOIs (SC7_L04-SC7_L09) after an initial scanning at the beginning, as the adjusted time slider shows (right).

the code snippet. Three participants incorrectly answered that the word to replace was *Sun*. The other five participants incorrectly answered *W*. If we look at these two participant groups, we find that the participants incorrectly answering *W* do not differ from those participants that answered correctly. However, the first group of participants (those incorrectly answering *Sun*) show different eye movement sequences. Participant P01 performed a one time linear reading of the source code, participant P12 focused on the important AOIS SC7_L07, SC7_L08, and SC7_L09 for less than 1 s, and participant P23 only briefly (<1.4 s) looked at the stimulus.

The other participants (both novices and non-novices) mainly focus on the relevant AOIs (SC7_L04-SC7_L10) after an initial scanning of the complete stimulus. Figure 5 shows an example of Participant P25 who after 1.3 s only focuses on AOIs SC7_L03 to SC7_L09.

5 DISCUSSION

We elaborate on results found using our qualitative and quantitative analysis and discuss them in the context of related work.

5.1 Reading of NT versus SC

Overall, we found weak evidence (cf. Figure 2A, top) that participants performed slightly better in answering the comprehension questions of the natural language texts as opposed to the source code snippets. Also, we found weak evidence that the completion time (cf. Figure 2B, top) for the natural language texts was lower than for the source code snippets. This might indicate that reading and comprehending natural language text is a bit easier and faster. Comparing the novices and non-novices for the source code, we found weak evidence for non-novices performing better and strong evidence that they took more time to comprehend and answer the source code snippets (cf. Figure 2A and B, bottom).

5.2 Coverage

We found strong evidence (cf. Figure 2C, top) that AOI coverage for natural language was higher. Interestingly, the AOI coverage for natural language texts was not 100%. If we assume that participants have to read each line of text this should be true. However, for the natural language text some participants did not focus on all AOIs.

For example, Participant P08 for three of the four natural language text stimuli (NT 2, NT 3, NT 10) did not focus on the last two AOIs. Inspecting the original raw gaze data for this participant, we see that this participant is wearing glasses and the eye movement data was not recorded correctly. However, this participant gave three correct answers for the natural language text stimuli. For NT 2 we find that participants P13 and P15 did not focus on the last AOI. However, this stimulus has only one word on the last line, which might mean that these two participants were able to read the last line using their peripheral vision or they just ignored the last word as not being relevant for understanding the task. Both participants answered the comprehension question correctly. For NT 3 only participant P23 has and for NT 10, participants P10, P13, P18 and P23 have not focused on all AOIs. In these cases, inaccuracies of the data recording seem to be the problem. Therefore, we can conclude that except for a few cases, participants read or focus on each line of a natural language text at least once.

This is different for the source code stimuli. The AOI coverage for both novices and non-novices was on average below 90%. However, the AOIs not focused on were not necessarily needed to answer the task correctly (e.g., include statements, closing brackets, functions headers, or return statements). We assume that participant with some experience in programming, learn that some parts of the code are not as relevant than other parts. Both novices and non-novices spent most of their dwell time on the actual and relevant source code, whereas the least amount of dwell time for both is on closing brackets and the include statement. This is in accordance with previous work, for example, Crosby et al. [2002; 1990] found that participants pay least attention to keywords and Busjahn et al. [2011] found keywords and numbers are least focused on.

5.3 Linear Reading Order

Overall, we found that participants read the natural language text mostly in linear order. This was especially true at the beginning, when participants started to read the text. However, the natural language text NT 10 is an exception. For this stimulus, the linear reading order is not as clear as for the other stimuli. This can have several reasons. First, all stimuli have the same size (1280 px × 720 px), but the first three stimuli have between 6 and 10 lines. NT 10, however, has 12 lines, which required that the font size had to be decreased and there was less space between lines to fit the text into the same area as the others. The decreased spacing leads to smaller AOIs, which may lead to more fixations being mismatched causing more jumping between consecutive AOIs.

5.4 Visual Analysis

The qualitative analysis using visualization techniques can help to extract common patterns that are hard to detect with a quantitative analysis alone. In addition, we could confirm some initial results and insights found with the qualitative analysis using a quantitative

analysis. For example, a first inspection of the transition graphs allowed us to immediately see that participants had not focused on some AOIs and we then calculated the AOI coverage to confirm that there was strong evidence for higher AOI coverage in NT texts.

To investigate the linear reading order we inspected the radial transition graphs and were able to find more linear reading for natural text stimuli than for source code snippets. Therefore, these results, the AOI coverage as well as the linear reading order, allows us to conclude that a qualitative and quantitative analysis can be used together to get the most insights from the eye movement data. With our analysis we showed that the visual analysis can be used as an initial analysis or more generally to form hypotheses, which can then be confirmed with quantitative results. The opposite is also true that a quantitative result can be explained through a visual inspection of the data, i.e., why a participant has a shorter completion time (e.g., participant P23 for SC 7) or a low AOI coverage (e.g., participants P13 and P15 for NT 2).

5.5 Threats to Validity

Similar to other studies we did not include comments [Busjahn et al. 2015; Sharafi et al. 2012; Uwano et al. 2006] and had to disguise method names [Aschwanden and Crosby 2006], for example, `doSomething` for SC 5 (cf. Figure 4). In programming, comments and proper names for methods, functions, and variables help to understand the functionality of source code. However, this might have caused participants not to read the actual source code and just the comments or method names.

6 CONCLUSIONS AND FUTURE WORK

We present results of an eye tracking study on natural language texts and short code snippets. We used a quantitative and qualitative approach to analyze the eye movement data. Using the radial transition graph visualization technique [Blascheck et al. 2017b] we found that natural language text is read in a more linear fashion and that participants focus more on important parts of source code. We were also able to extract common patterns, analyze different reading behavior, and explore general strategies of participants using the radial transition graph. To further analyze the eye movement data, some extensions such as clustering of participants would be valuable. Also, integrating more data sources (e.g., retrospective interviews) would help to confirm our assumptions. Furthermore, we believe visualization techniques can benefit eye tracking researchers to explain their quantitative results by providing more meaningful explanations.

ACKNOWLEDGMENTS

We are grateful to the participants who took part in this study. We thank Lonni Besançon for providing R code (<https://aviz.fr/ci/>) to calculate the 95% Bootstrap confidence intervals. This work is supported in part by grants from the National Science Foundation under grant numbers CCF 18-55756 and CCF 15-53573.

REFERENCES

Christoph Aschwanden and Martha Crosby. 2006. Code scanning patterns in program comprehension. In *Proceedings of the Hawaii International Conference on System Sciences*. 1–10.

- Roman Bednarik. 2012. Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human-Computer Studies* 70, 2 (2012), 143–155.
- Lonni Besançon and Pierre Dragicevic. 2017. La Différence Significative entre Valeurs p et Intervalles de Confiance (The Significant Difference between p-values and Confidence Intervals). In *Conférence Francophone sur l'Interaction Homme-Machine*.
- Tanja Blascheck, Kuno Kurzhals, Michael Raschke, Michael Burch, Daniel Weiskopf, and Thomas Ertl. 2017a. Visualization of Eye Tracking Data: A Taxonomy and Survey. *Computer Graphics Forum* 36, 8 (2017), 260–284.
- Tanja Blascheck, Markus Schweizer, Fabian Beck, and Thomas Ertl. 2017b. Visual Comparison of Eye Movement Patterns. *Computer Graphics Forum* 36, 3 (2017), 87–97.
- Ruven Brooks. 1983. Towards a theory of the comprehension of computer programs. *International journal of man-machine studies* 18, 6 (1983), 543–554.
- Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *Proceedings of International Conference on Program Comprehension*. IEEE Computer Society Press, 255–265.
- Teresa Busjahn, Roman Bednarik, and Carsten Schulte. 2014a. What influences dwell time during source code reading? Analysis of element type and frequency as factors. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 335–338.
- Teresa Busjahn, Carsten Schulte, and Andreas Busjahn. 2011. Analysis of code reading to gain more insight in program comprehension. In *Proceedings of the Koli Calling International Conference on Computing Education Research*. ACM, 1–9.
- Teresa Busjahn, Carsten Schulte, Bonita Sharif, Begel Andrew, Michael Hansen, Roman Bednarik, Paul Orlov, Petri Ihantola, Galina Shchekotova, and Maria Antropova. 2014b. Eye tracking in computing education. In *Proceedings of the Annual Conference on International Computing Education Research*. ACM, 3–10.
- Benjamin Clark and Bonita Sharif. 2017. iTraceVis: Visualizing Eye Movement Data Within Eclipse. In *IEEE Working Conference on Software Visualization*. IEEE Computer Society Press, 22–32.
- Martha Crosby, Jean Scholtz, and Susan Wiedenbeck. 2002. The roles Beacons play in comprehension for novice and expert programmers. In *Proceedings of the Workshop of the Psychology of Programming Interest Group*. 58–73.
- Martha Crosby and Jan Stelovsky. 1990. How do we read algorithms? A case study. *Computer* 23, 1 (1990), 25–35.
- Geoff Cumming. 2013. *Understanding the New Statistics: Effect Sizes, Confidence Intervals, and Meta-analysis* (1st ed.). Routledge.
- Pierre Dragicevic. 2016. Fair Statistical Communication in HCI. In *Modern Statistical Methods for HCI*, Judy Robertson and Maurits Kaptein (Eds.). Springer, 291–330.
- Gerd Gigerenzer. 2004. Mindless statistics. *The Journal of Socio-Economics* 33, 5 (2004), 587–606.
- Steven N Goodman. 1999. Toward evidence-based medical statistics. 1: The P value fallacy. *Annals of Internal Medicine* 130, 12 (1999), 995–1004. <https://doi.org/10.7326/0003-4819-130-12-199906150-00008>
- Drew T. Guarnera, Corey A. Bryant, Ashwin Mishra, Jonathan I. Maletic, and Bonita Sharif. 2018. iTrace: eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*. ETRA 2018, Warsaw, Poland, June 14-17, 2018. 105:1–105:3.
- Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost Van de Weijer. 2011. *Eye Tracking: A Comprehensive Guide to Methods and Measures* (1 ed.). Oxford University Press.
- Martin Krzywinski and Naomi Altman. 2013. Points of Significance: Error bars. *Nature Methods* 10, 10 (2013), 921–922.
- S. Letovsky and E. Soloway. 1986. Delocalized Plans and Program Comprehension. *IEEE Softw.* 3, 3 (1986), 41–49.
- Unaizah Obaidallah, Mohammed Al Haek, and Peter C.-H. Cheng. 2018. A Survey on the Usage of Eye-Tracking in Computer Programming. *Comput. Surveys* 51, 1 (2018), 5:1–5:58.
- Christopher Palmer and Bonita Sharif. 2016. Towards automating fixation correction for source code. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, ETRA 2016, Charleston, SC, USA, March 14-17, 2016*. 65–68.
- Frank Schmidt and John Hunter. 1997. Eight common but false objections to the discontinuation of significance testing in the analysis of research data. In *What If There Were No Significance Tests?*, Lisa Harlow, Stanley Mulaik, and James Steiger (Eds.). Lawrence Erlbaum Associates, 37–64.
- Timothy Shaffer, Jenna Wise, Braden Walters, Sebastian Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks. In *Proceedings of the Joint Meeting on Foundations of Software Engineering*. ACM, 954–957.
- Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gaël Guéhéneuc. 2015. Eye-Tracking Metrics in Software Engineering. In *Asia-Pacific Software Engineering Conference*. IEEE Computer Society Press, 96–103.
- Zohreh Sharafi, Zéphyrin Soh, Yann-Gaël Guéhéneuc, and Giuliano Antoniol. 2012. Women and men – Different but equal: On the impact of identifier style on source code reading. In *Proceedings of International Conference on Program Comprehension*. IEEE Computer Society Press, 27–36.
- Bonita Sharif, Michael Falcone, and Jonathan Maletic. 2012. An eye-tracking study on the role of scan time in finding source code defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 381–384.
- Oleg Špakov, Harri Siirtola, Howell Istance, and Kari-Jouko Räihä. 2017. Visualizing the Reading Activity of People Learning to Read. *Journal of Eye Movement Research* 10, 5 (2017), 1–12.
- Rachel Turner, Michael Falcone, Bonita Sharif, and Alina Lazar. 2014. An eye-tracking study assessing the comprehension of C++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 231–234.
- Hidetake Uwano, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. 2006. Analyzing individual performance of source code review using reviewers' eye movement. In *Proceedings of the Symposium on Eye Tracking Research and Applications*. ACM, 133–140.
- Hidetake Uwano, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. 2007. Exploiting eye movements for evaluating reviewer's performance in software review. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* 90, 10 (2007), 2290–2300.
- Chia-Kai Yang and Chat Wacharamanatham. 2018. Alpscarf: Augmenting Scarf Plots for Exploring Temporal Gaze Patterns. In *Proceedings of the CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, 503:1–503:6.