Visually Analyzing Students' Gaze on C++ Code Snippets

Cole S Peterson, Jonathan A Saddler, Tanja Blascheck, Bonita Sharif

Visually Analyzing Students' Gaze on C++ Code Snippets

Cole S. Peterson*, Jonathan A. Saddler*, Tanja Blascheck† and Bonita Sharif*
*Department of Computer Science, and Engineering, University of Nebraska-Lincoln, Lincoln, Nebraska, USA 68588
†Inria, Saclay, France
Emails: cole.scott.peterson@huskers.unl.edu, saddler@huskers.unl.edu, tanja.blascheck@inria.fr, bsharif@unl.edu

***Abstract*—The paper presents an eye tracking study with 17 students (12 novices, 5 non-novices) reading C++ methods. The novices were students who participated in the study during the last week of their semester learning C++. The non-novices were senior students who had been exposed to programming before. We report on the reading behavior of three C++ methods that use different language constructs. We analyze fixations at the line level of the code using visualizations to derive insights into code reading. Results show that most transitions were made to code lines that are close to the current line read. We observe that a large percentage of the total fixation duration is made on a small number of lines and that related lines are often viewed together in a series of short fixations.**

***Index Terms*—eye tracking, visual analysis, program comprehension, C++ source code, scanpaths, transitions**

I. Introduction

Program comprehension is a subarea of software engineering with a history of contributions towards theory and experiments that bring us closer to understanding how programmers actually read and understand code [1]. However, we still do not have a theory that can completely model how novices and experts read code. Reading code efficiently is an essential skill to have. It is a precursor to many software engineering activities such as bug fixing, code reviewing, or new feature additions. Since 1990 and more so after 2006, eye tracking researchers in software engineering have been working towards conducting studies to build on the body of knowledge of program comprehension in a systematic way [2], [3]. We only expect this trend to continue as eye trackers get more accessible and are more suitable for the community to adopt as support infrastructure [4] improves.

In this paper, we present an eye tracking study conducted with three short C++ code snippets at the end of a semester long programming course. Because eye movement data appears to have many individual differences [5] it is sometimes difficult to quantitatively assess significance of differences between eye movements. Eye movement data is a signal produced over time as the participant progresses with a task. Understanding and analyzing similarities and differences of eye movement data over time is not as easy as using aggregate measurements such as the total number of fixations and their corresponding durations. To make it easier to find trends and patterns in the data, we believe that visualizations should be used to quantify patterns in time and space. However, there are not many visualization tools available for eye tracking researchers to use. To bridge this gap, we leverage two visualization techniques namely radial transition graphs [6] and parallel scanpaths [7] introduced in prior work to help us analyze the data from the eye tracking study presented here and show the viability of using visualizations for eye tracking analysis in program comprehension.

The research questions we seek to answer in this paper are:

- RQ1: How do students transition their gaze between different lines of short code snippets?
- RQ2: How can we visually compare differences in scanpath patterns between several students?

The main contribution of this paper is using existing visualization techniques to qualitatively explain transitions between source code lines and comparing scanpaths among various participants over time.

II. Related Work

In this section, we present selected related work on program comprehension and eye tracking visualizations.

Program Comprehension Studies. Busjahn et al. [8] introduce linearity metrics based on fixation counts and saccades 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 way when compared to novices. They also found novices read source code less linearly than natural language text. Crosby et al. [9] were one of the first researchers to study comprehension using an eye tracker in 1990. They used fill in the blank tasks in Pascal programs to determine program comprehension. Rodeghero et al. [10] report on reading patterns of programmers during summarization tasks. They found that on average programmers read from top to bottom 49% of the time. Begel et al. [11] report on eye movements during code review and attempt to classify which code elements trigger deliberation and how long reviewers take to verify their hypotheses. They also report on code skimming vs. careful reading. However, none of the above mentioned studies use visualizations in a systematic manner to help with comparisons between participants' transitions between areas of interest (AOIs) and scanpaths.

Eye Tracking Visualizations. There is not much work in the area of visualizing eye movement data. Blascheck et al. [12] are one of the few researchers that focus on various aspects of visualizing eye movements. Researchers can use visualizations of eye movement data to build and verify hypotheses in

addition to a more quantitative approach. Spakov et al. [13] developed another approach to analyze reading activity of people. They integrate several visualization techniques to analyze eye movement data collected in a reading study. Clark et al. [14] present iTraceVis which is built into the Eclipse IDE and based on iTrace [4]. These two visualizations are not conducive to compare multiple participants because they do not compare transitions between AOIs and scanpaths visually.

In this paper, we use state-of-the-art visualization techniques to analyze our eye movement data. We compare participants' line viewing behavior using radial transition graphs [6] and parallel scanpaths [7] to inspect the eye movement data over time. The radial transition graphs allow us to investigate the distribution of fixation duration for multiple lines as well as inspect transitions between individual lines in a compact manner. In contrast, the parallel scanpaths represent the sequence of transitions in a linear fashion, keeping the order of the source code lines, to find common lines that are focused on the most and allow us to see specific reading patterns.

III. Study Design

The goal of this study is to determine how students introduced to C++ programming read and comprehend programs. In particular, we are interested in how students read and transition between lines and how they compare with each other in terms of the line reading patterns.

A. Participants

We recruited a total of 17 students to participate in this study. Of these, 12 students were enrolled in an *Introduction to C++* course. We label these 12 students as novices because prior to this class they had not been exposed to programming. The remaining 5 students were considered non-novices. They were also students but were exposed to programming before. These two groups, novices and non-novices were determined by the number of years each participant completed in the university's 4-year program and graduate program. Those students in the graduate program, along with those who completed 3 of 4 years of the undergraduate program, were given the title non-novices to separate them from the novices with less than 3 years of experience in institutionalized coding. The novices in our study had not taken any programming prior to the C++ class they were investigated in.

B. Stimuli and Comprehension Tasks

We investigated three C++ programs, which are shown in Fig. 1, 2, and 3. Participants were given as much time as needed to read each of the three programs. After each program, participants were assigned one of three randomly chosen comprehension questions: a question about the program's output, a short answer question, or a multiple choice question. The accuracy of answers to these questions is not within the scope of this paper. We leave this as part of our ongoing future analysis.

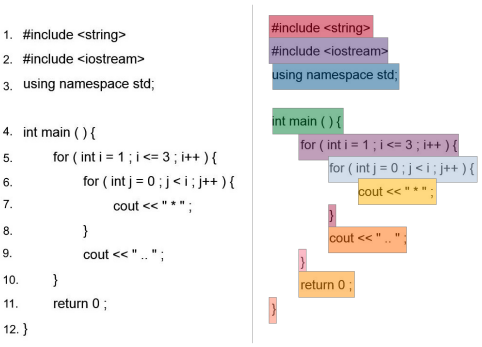

Fig. 1. PrintPatternR program, which prints a pattern of stars and dots (left) and the line-level AOIs used in our analysis to map fixations to lines (right).

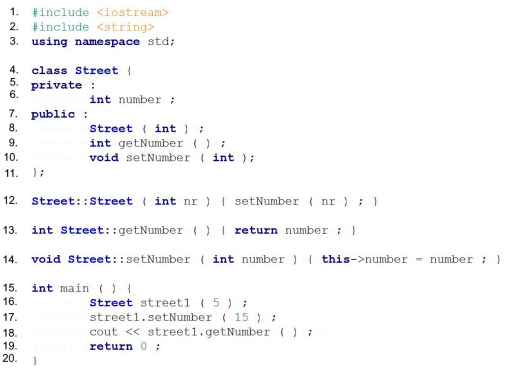

Fig. 2. StreetH program, which creates and sets an integer field value of class object Street. The AOIs are not shown, however, they are similar to Fig. 1.

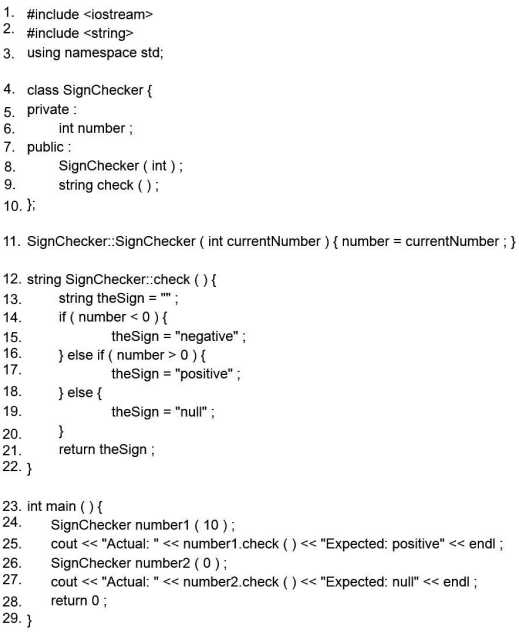

Fig. 3. SignCheckerClassMR program, which returns the sign of an integer passed to the SignChecker constructor.

C. Eye Tracking Apparatus

We used the Tobii X60 eye tracker with the Tobii Studio environment that supports recording of eye movement data on images and AOI mapping of fixations to lines. Each of the programs was shown as an image (using Tobii Studio) on the screen, after which a comprehension question appeared via a Google form for the participants to answer. We ran a fixation filter (IV-T) on the raw gazes. All analyses conducted are on the fixations the filter generated.

D. Procedure

On the day of the study, participants first signed an informed consent form and were given a pre-questionnaire to gauge their level of expertise. After they read initial instructions on what was required of them, they began the study. They were shown each program for as long as they needed until they clicked next, at which point they were required to answer a comprehension question via a Google form. They did this for all three programs. Note, that the study was done in the last week of the semester. The class instructor of the C++ class (not a paper author) had covered all concepts in the programs.

E. Data Correction and Mapping

Steps were taken to engineer a consistent dataset of fixations that properly capture what participants were viewing throughout the study. We used an in-house tool, *Vizmanip*, to visually locate strands of ten fixations that had drifted away from the program's line. Two correctors had to agree that the group of ten fixations needed correction before they were adjusted. We needed to account for drift that occurs during an eye tracking study. The source code of the tool we used to apply corrections to our data can be found at https://github.com/SERESLab/fixation-correction-vizmanip.

Even though Tobii Studio allows us to map fixations to lines, it does not allow us to correct fixations. For this purpose, we needed to export all the data out of Tobii Studio, correct the data using our Vizmanip tool and map lines to fixations using another tool named *eyeCode* (https://github.com/synesthesiam/eyecode), a Python data-processing library. We used *eyeCode* to map our corrected fixations to their line-numbered locations in each program. We used eyeCode's special AOI data format to encode the information. The software also comes with a special AOI image recognition tool that can be used to automatically generate AOIs for source code snippets. We had to make a few consistent modifications to the process to get this mechanism parsing to behave properly, such as consistently ensuring a set amount of padding (12 px to top and bottom) of every returned AOI to account for vertical line gaps. In about six cases, the authors were forced to manually create AOIs because the generation algorithm failed to provide the correct AOIs for the lines in the program. Three authors verified the AOIs over the course of four total hours.

F. Threats to Validity

The eye movement data was corrected manually to ensure mapping to lines was accurate. To mitigate any issues with this process, both authors performing the corrections had to agree before moving fixations. If no agreement could be reached, the points were not moved. In addition, corrections were not cherry-picked fixations. The fixations were moved only if they followed a trajectory similar to the source code line near it and always in groups of ten fixations per line.

The authors set the difficulty rating of programs based on the types of constructs used. For example, the SignCheckerClassMR program and PrintPatternR program were considered to be difficult because they involved nested if's and nested for-loops. In addition, one of the programs (StreetH) was used with syntax highlighting (in Courier New font) with the rest being black and white shown in Arial font. This was because the three programs analyzed in this paper were part of a larger ongoing study with different research questions. We do not consider syntax highlighting to be a factor in the current study because we only directly compare eye movements on the same code snippet to each other.

Another possible threat could be the programs used because it is possible that students might not be familiar with the concepts. We argue, however, that all constructs shown in the programs were covered multiple times during the course, the novice participants attended, which means students should have had time to get acquainted with those constructs.

IV. Study Results

We report on the results to each of the two research questions posted in the introduction. Of the 17 participants, we gathered viable eye movement data from 15 participants. We use lines as the unit of analysis because a transition between lines can be represented by a change in height which lends itself to visualizations.

A. RQ1 Results: Transitions Between Lines

Before we analyze eye movement patterns using visualizations, it is helpful to understand the meaning of the visualizations used and how they are constructed. In the following section, we analyze two types of visualizations which are called distribution graphs and transition graphs. For both types of visualizations, the colored sectors are arranged in a clockwise manner. The first line is always the top most sector and subsequent lines are placed clockwise. Lines that are next to each other in the program, for example, line 6 and 7, are placed next to each other in the visualization graphs. Some sectors in the graph are represented by the same color because many lines were used, however they can still be distinguished by remembering that the sectors are arranged sequentially and clockwise and looking at the previous and subsequent sectors.

The distribution graphs have the sectors take up an amount of space proportional to the amount of total duration of fixations on the line. The transition graphs have lines connecting different sectors. The black dot on the sector represents outgoing transitions and the white dot on the sector represents incoming transitions. The width of the transition line is based on the frequency of the transition from one line to another with a more frequent transition resulting in

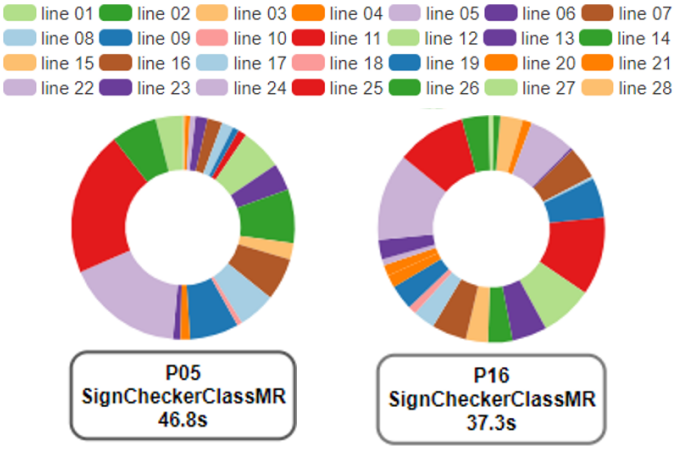

Fig. 4. Comparison of distributions of fixations on lines based on duration between P05, a non-novice (shown on the left), and P16, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

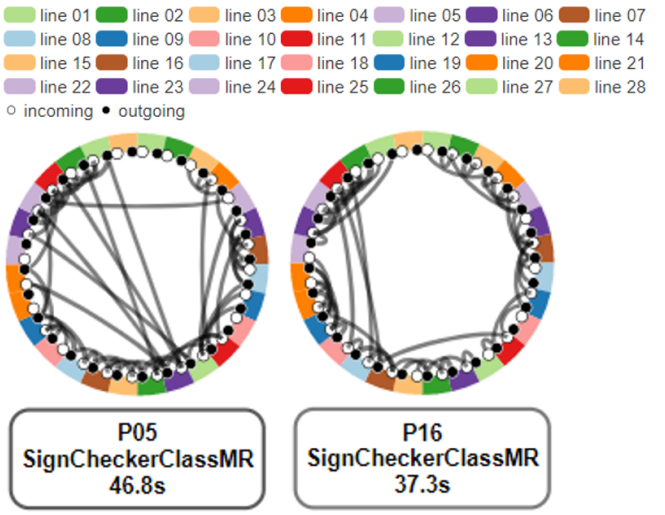

Fig. 5. Comparison of transitions between lines for P05, a non-novice (shown on the left), and P16, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

a thicker line. These graphs are generated via the online tool at http://www.rtgct.fbeck.com/.

We chose this radial representation despite its drawback of placing the first and last line of the source code next to each other because it allows us to analyze the transitions between AOIs. For example, the graphs show if the transitions are mostly from lines that are next or close to each other (e.g., Fig. 5 right) in contrast to transitions that go across the complete graph (e.g., Fig. 5 left), which indicates that the transitions span a long range of line in the source code stimulus. This gives us first hints about the reading behavior of participants and which different reading strategies are present.

SignCheckerClassMR. One component of eye movement analysis is the distribution of time spent in various AOIs. In Fig. 4, we can see the distribution of fixation duration over the lines in SignCheckerClassMR between two participants P05, a non-novice, and P16, a novice. We can see that P05 looked more at line 24 and line 25 than P16. In fact, we can see that the main method's body from line 24 to 27 take up almost half of the total fixation duration of P05, whereas it takes only a quarter of the total fixation duration of P16.

Another large component of eye movement analysis is investigating transitions between different AOIs, in this case lines. We can see a visual representation of these transitions in Fig. 5. If we examine P16, we can see that most transitions are to a line that is relatively close by. The transitions that occur to lines farther away pass closer to the center of the circle. In P16's transition graph, we can see transitions occurring between line 16 and line 25 and between line 25 and line 17. In contrast, P05's transition graph shows us that while they still read the code in a subsequent manner (e.g., the chain of transitions between line 11 and line 21), they have more transitions between lines of code that are farther away. We can see that P05's transition graph has more lines in the center indicating that several transitions occurred between lines that are at opposite ends of the graph, which means that the transition spanned up to half of the lines in the program.

This representation makes the last and first line appear next to each other. Because of this factor, transitions between the top section and bottom section of the code may appear like transitions that are between lines that are close together. An example of this would be the transition from line 24 to line 5 in P05's transition graph, which appear like a transition over a smaller span than the nearly $2/3$ of the program's lines that the transition occurs over. Knowing this pattern, we have to pay closer attention to transitions that occur between the top left and top right quarters of the transition graphs.

PrintPatternR. This is a short and simple code snippet relative to the other snippets that we analyze. Fig. 6 shows the distribution of fixations on lines based on the duration for PrintPatternR. We can see that only a few lines make up the majority of the fixation duration for both P06 and P17. Line 6 and 7, the inner for-block and print statement, are the most viewed lines in the PrintPatternR. We can see that P06 did not look at many lines outside of line 6 and 7. They looked at the outer for-loop (line 5) and the second print statement (line 9) for a small amount of time and the rest of the lines are barely visible on the chart indicating that these four lines were almost all that was needed to understand the program. We can see these trends for P17 to a lesser extent. It appears that line 6 and 7 made up a majority of the fixation duration for P17, but that this participant fixated on line 3 and 4 much more than P06. These lines, the using namespace and main method signature, are boilerplate and do not provide any additional information about the PrintPatternR.

The transition graph for PrintPatternR can be seen in Fig. 7. We can see that the transitions between the lines are fairly similar between these two participants. There is a large amount of transitions between line 5, 6, and 7 for both P06 and P07. While P06 has more transitions to lines that are farther away, it

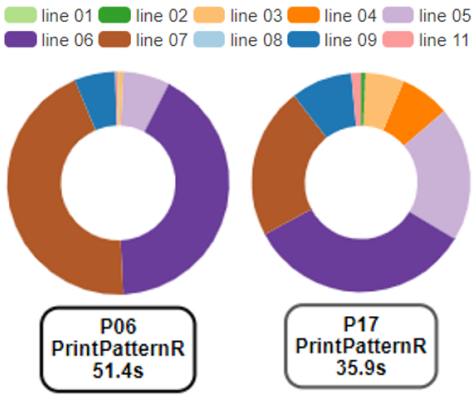

Fig. 6. Comparison of distributions of fixations on lines based on duration between P06, a non-novice (shown on the left), and P17, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

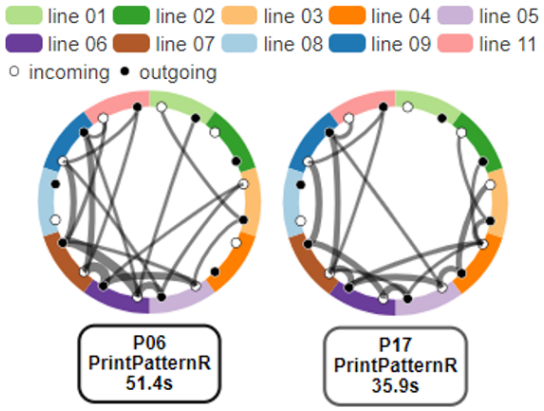

Fig. 7. Comparison of transitions between lines for P06, a non-novice (shown on the left), and P17, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

is not as stark of a difference as seen in SignCheckerClassMR. We can see that there are no transitions to or from both line 6 and 8 for P06, which supports the distribution of fixation duration we saw in the previous paragraph. P06 and P17 did not look at the first few lines of PrintPatternR, which have only some amount of transitions between these earlier lines.

StreetH. This program is a longer program than PrintPatternR. We can see the distribution graph of StreetH in Fig. 8. We presented StreetH to the participants with syntax highlighting and unlike the previous two examples, there is not a clearly visible similarity between the two participant's fixation duration distributions. However, we can see there are common lines that both participants fixated on. Some of these more common lines are lines 5 and 7 and lines 12 to 16. Lines 5 and 7 correspond to the private and public access modifiers in the Street class respectively. Line 12 to 14 correspond to the various constructors and method implementations for the Street class. Line 15 corresponds to the main method signature and line 16 corresponds to the initialization of a variable of type Street. Of these groups, the fixations on line 12, 13, and 14 corresponding to the implementations of methods and

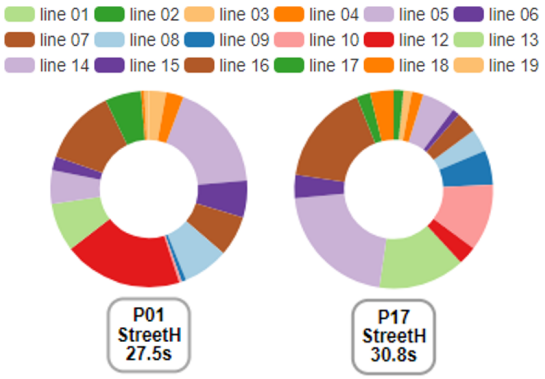

Fig. 8. Comparison of distributions of fixations on lines based on duration between P01, a non-novice (shown on the left), and P17, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

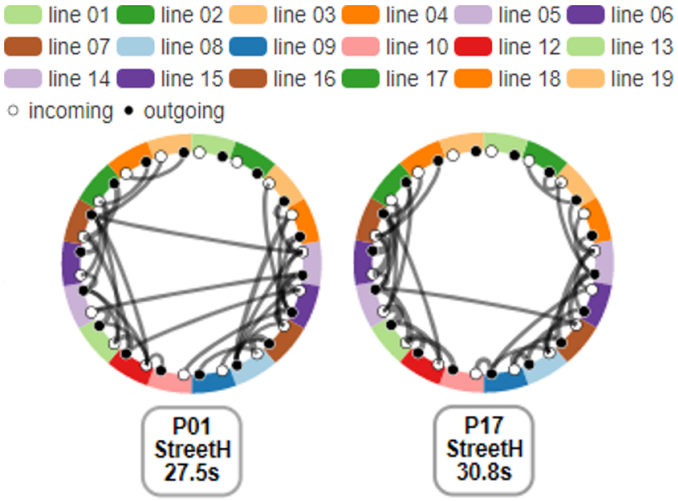

Fig. 9. Comparison of transitions between lines for P01, a non-novice (shown on the left), and P17, a novice (shown on the right), with line 1 at the top of the graph and subsequent lines being placed in a clockwise manner.

constructors for the Street class account for over a quarter of the total fixation duration for both P01 and P17.

Looking at the transition graph for these two participants in Fig. 9, we can see that only a few transitions occurred between the top half of the program including the boilerplate and class definitions and the bottom half of the program including the method implementations and main method. P01 has more of these transitions but does not have as many transitions spanning a large amount of lines as was seen in some of the transition graphs for SignCheckerClassMR. While P01 does not have many transitions spanning a large amount of the program, the remaining transitions are not all transitions to consecutive lines. There is still a number of transitions to lines that are several lines away. We can see that P17 does not have as many transitions made to the lines that are further away. The transition graph for this program looks similar to P16's transition graph for SignCheckerClassMR with regards to the transitions made in the first half of the program that appear to

be made to lines that were close together. Even though StreetH had syntax highlighting, we did not observe overall trends for duration distribution or transitions that differed from the other two programs at the line level.

B. RQ2 Results: Parallel Scanpath Comparisons

The parallel scanpath visualization (Fig. 10) shows time on the y-axis and AOIs on the x-axis. The scanpath is then shown as a line through the different AOIs. This visualization technique allows us to compare the different completion times, i.e., which participants were faster or took longer, which AOIs were focused on when, as well as finding specific patterns (set of lines highlighted in varied colors in Fig. 10).

PrintPatternR. Fig. 10 shows the parallel scanpath for all participants for PrintPatternR. In this comparison we can see that all participants focused at least once on lines 5 to 8 (highlighted parts in Fig. 10), which are the important lines to understand what this program is doing. However, we can also observe that some participants also focused on other parts of the program, which are not so relevant (e.g., P05, P10, P11, P17). The green (lines 5 and 6) and purple highlights (line 7) show the areas in PrintPatternR that many participants fixated on the most in all our examples. These lines correspond to the inner (line 5) and outer (line 6) control loops and their content. We can see that most participants fixated on the inner and outer for-loops (lines 5 and 6, green) closer to the beginning of the session. For many, it is the first important line that they fixated on and was often followed by fixating on the first print statement on line 7 (purple). While we also see many fixations on line 9 (yellow), the second print statement, they seem secondary to line 7 because line 9 is not focused on as frequently and is often between other fixations of line 7 which indicate that it is normally being viewed within the context of the first print statement.

StreetH. For the StreetH program, we find similar behavior (see Fig. 11). The main method (lines 16 to 19, pink) received few fixations. Instead we see the method and class declarations make up most of the fixation time. We can see that the constructor (line 12, purple), the getNumber method (line 13, yellow), and the setNumber method (line 14, orange) received more attention. We see that these three related lines are viewed close together. For most participants, they looked at the constructor (line 12, purple) first, then the getNumber method (line 13, yellow), and then transitioned to the setNumber method (line 14, orange). However, these are clean transitions. Often there is a period of transitions between two of these lines. The method prototypes (line 8 to 10, green) are often viewed separately from the implementations of the methods. They were most often fixated on toward the beginning of the session and before the participants looked at the implementations of the methods on lines 12 to 14. We also see in this visualization that most transitions made are to lines that are close to the current line. While we see some sharp transitions covering a large amount of distance in the x-axis, which show that a large amount of lines were covered, the more common type of transitions are to lines that are only a few lines away. While StreetH has syntax highlighting, we did not observe any noticeable differences in trends in the participant's scanpaths compared to the other two files.

SignCheckerClassMR. In Fig. 12, we see the scanpaths of SignCheckerClassMR for all participants. The first observation we see is that the if statements (line 14 to 20, purple) are only viewed for a short time, but they are viewed together without a large number of transitions. We also see that the scanpath inside these highlighted areas of the if block is in an almost straight line indicating that participants read the lines roughly from the top of the chunk to the bottom of the chunk. We also see that there was little interaction between the lines we highlighted: the constructor (line 11, green), the first usage of the constructor (line 24, yellow), and the second usage of the constructor (line 26, orange). While there are still short transitions made to and from the lines we highlight, the other line involved in the transition were ones that we did not highlight. Similar to the previous scanpaths of StreetH, we observe that most transitions were made to lines that were close by. We see only few sharp lines in the scanpath that represent transitions to lines that were far away, which is consistent with what is seen in StreetH.

V. Discussion

We found several common observations between the three programs about the lines being fixated on and the differences and similarities between novices and non-novices. The first observation is that boilerplate code such as includes and the using namespace are not focused on as much as the other lines in the code snippets. We also observed that a large percentage of fixation duration is distributed over a small number of lines with this being more pronounced for the non-novices we analyzed. The final observation is that most transitions were to lines that were close to the current line being fixated on. While non-novices have more transitions that span multiple lines, most of their transitions were to lines that were close.

When analyzing the parallel scanpaths of multiple participants we made several common observations. We found that the scanpaths showed the same trend that most transitions were to lines that were close to the current line. We can see sharp lines representing transitions to lines farther away break up periods of transitions over shorter distances. We also observed that the single lines we highlighted were involved in series of short transitions at some point. However, when we grouped several related lines together we noticed they were less often a part of a series of short transitions. This hints that developers fixate on code not based on lines alone, but on chunks of related lines. We were able to observe these trends purely through analysis of several visualizations. We were able to easily compare pairs of participants and find common trends and observations using existing visualizations technqiues.

VI. Conclusions and Future Work

In this paper we present a visual analysis of eye movement data on three short C++ code snippets. Two kinds of visualizations were used leveraging different aspects of the eye

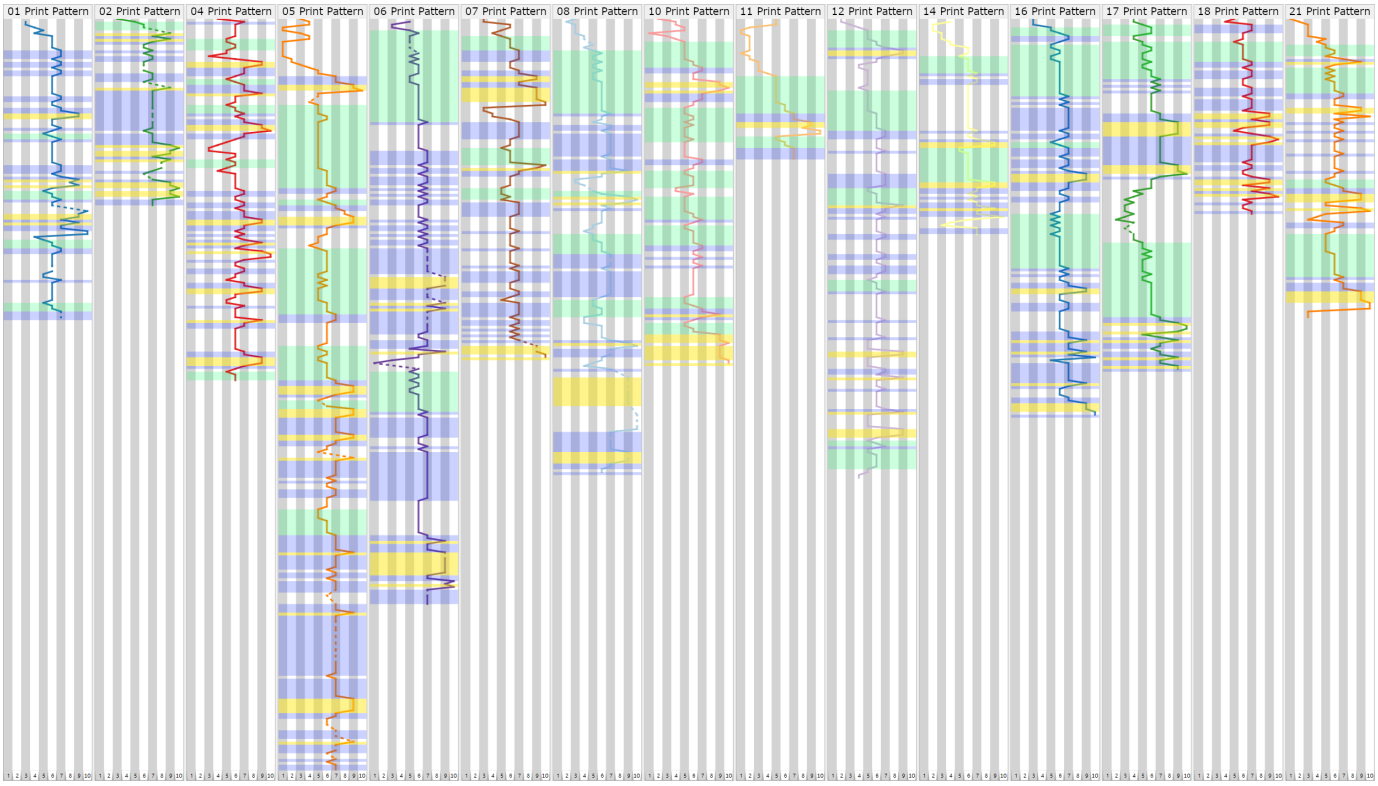

Fig. 10. Parallel scanpath visualization of PrintPatternR for all participants. We have highlighted in green lines 5 and 6, in purple line 7, and in yellow line 9. These highlights shown when participants focused on these specific lines.

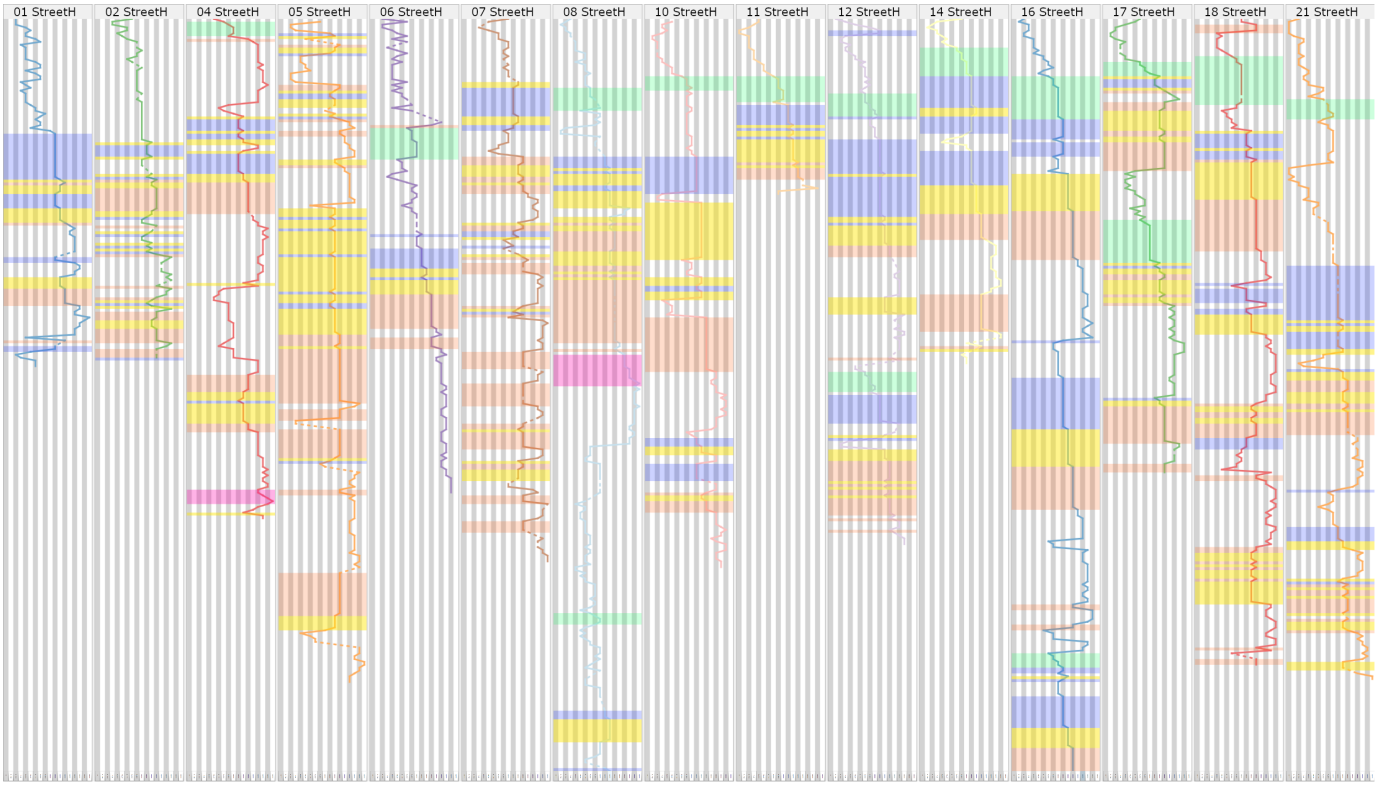

Fig. 11. Parallel scanpath visualization of StreetH for all participants. We have highlighted in green lines 8-10, in purple line 12, in yellow line 13, in orange line 14, and in pink lines 16 to 19. These highlights shown when participants focused on these specific lines.

Fig. 12. Parallel scanpath visualization of SignCheckerClassMR for all participants. We have highlighted in green line 11, in purple lines 14-20, in yellow line 24, and in orange line 26. These highlights shown when participants focused on these specific lines.

movement data. We found that most transitions between lines in code were to lines that were close to the current line, a large percentage of fixation duration is distributed over a small number of lines, and related lines are viewed together and often in a series of short fixations. As part of future work, we plan on extending such visualizations to incorporate fixations generated on larger code snippets encompassing more methods and files. In addition, we plan on analyzing line chunks that could resemble beacons to see if other trends are noticeable between chunks.

Acknowledgment

We are grateful to all the participants who took part in this study. This work is supported in part by grants from the National Science Foundation under grant numbers CCF 18-55756 and CCF 15-53573.

References

[1] M. D. Storey, "Theories, tools and research methods in program comprehension: past, present and future," *Software Quality Journal*, vol. 14, no. 3, pp. 187–208, 2006. [Online]. Available: https://doi.org/10.1007/s11219-006-9216-4

[2] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng, "A survey on the usage of eye-tracking in computer programming," *ACM Comput. Surv.*, vol. 51, no. 1, pp. 5:1–5:58, Jan. 2018.

[3] Z. Sharafi, Z. Soh, and Y.-G. Guéhéneuc, "A systematic literature review on the usage of eye-tracking in software engineering," *Information and Software Technology (IST)*, 2015.

[4] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, "itrace: Eye tracking infrastructure for development environments," in *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, ser. ETRA '18, 2018, pp. 105:1–105:3.

[5] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," in *Proceedings of the 2006 symposium on Eye tracking research & applications*, ser. ETRA '06. ACM, 2006, pp. 133–140.

[6] T. Blascheck, M. Schweizer, F. Beck, and T. Ertl, "Visual comparison of eye movement patterns," *Computer Graphics Forum*, vol. 36, no. 3, pp. 87–97, 2017.

[7] M. Raschke, D. Herr, T. Blascheck, M. Burch, M. Schrauf, S. Willmann, and T. Ertl, "A visual approach for scan path comparison," in *Proceedings of ETRA*. ACM, 2014, pp. 135–142.

[8] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye movements in code reading: Relaxing the linear order," in *Proceedings of International Conference on Program Comprehension*. IEEE Computer Society Press, 2015, pp. 255–265.

[9] M. Crosby and J. Stelovsky, "How do we read algorithms? A case study," *Computer*, vol. 23, no. 1, pp. 25–35, 1990.

[10] P. Rodeghero and C. McMillan, "An empirical study on the patterns of eye movement during summarization tasks," in *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, vol. 00, 2015, pp. 1–10.

[11] A. Begel and H. Vrzakova, "Eye movements in code review," in *Proceedings of the Workshop on Eye Movements in Programming*, ser. EMIP '18, 2018, pp. 5:1–5:5.

[12] T. Blascheck, K. Kurzhals, M. Raschke, M. Burch, D. Weiskopf, and T. Ertl, "Visualization of eye tracking data: A taxonomy and survey," *Computer Graphics Forum*, vol. 36, no. 8, pp. 260–284, 2017.

[13] O. Špakov, H. Siirtola, H. Istance, and K.-J. Räihä, "Visualizing the reading activity of people learning to read," *Journal of Eye Movement Research*, vol. 10, no. 5, pp. 1–12, 2017.

[14] B. Clark and B. Sharif, "iTraceVis: Visualizing eye movement data within eclipse," in *IEEE Working Conference on Software Visualization*. IEEE Computer Society Press, 2017, pp. 22–32.