Analysis of Static Code Reading Patterns

Christine Lourrine S. Tablatin
Department of Information Systems and Computer Science
Ateneo de Manila University
Philippines
tablatinchristine@gmail.com

Ma. Mercedes T. Rodrigo
Department of Information Systems and Computer Science
Ateneo de Manila University
Philippines
mrodrigo@ateneo.edu

ABSTRACT

Code reading is a prerequisite of program comprehension and is defined as a process of visually perceiving the source code with the focus of understanding how the code works as well as finding bugs and defects. Strategies employed on code reading affect a programmer's success rate of the comprehension tasks. However, little is known about the code reading patterns or visual strategies used by the students during programming tasks. We study how high performing and low performing students read source code while performing a static code reading task of identifying bugs and use eye tracking data to reveal their common code reading patterns. Results showed that high performing students spend less fixation durations and have less number of fixations on the locations of code errors compared to low performing students. A common scanpath was generated which describes the code reading patterns employed by students of different skill levels. This is a preliminary analysis and further studies with larger data sample would be conducted to be able to validate and generalize these results.

CCS CONCEPTS

• General and reference → Evaluation

Keywords

Eye tracking; code reading; code reading patterns; scanpath; Scanpath Trend Analysis

1.INTRODUCTION

Program comprehension is an integral aspect of software development and maintenance since programmers spend a lot of their time comprehending source code [18]. According to [3], source code is a structured document which can be difficult to comprehend. The importance of source code comprehension in developing and maintaining software, and the difficulty experienced by programmers in comprehending source code, offer opportunities for researchers to study how programmers think when performing program comprehension tasks [2, 3, 19, 25]. To gain insights into the behavior and strategies programmers exhibit during the comprehension process, researchers used techniques such as think-aloud protocols, observational studies, and eye tracking [2].

Code reading is a prerequisite of program comprehension. It is a process of perceiving the source code visually with the focus of understanding how the code works as well as finding bugs and defects [4, 6]. Analyzing what the programmer looks at while reading source code can be monitored using an eye tracker. Tracking the subject's eye movements over the stimulus can show their focus of attention which makes it possible to draw conclusions about their underlying cognitive processes and describe the actual comprehension strategies used [1, 8].

Recent studies were conducted using eye tracking technology to uncover programmers' cognitive processes while performing tasks such as program comprehension [5, 12, 15, 17, 19]. While code reading is considered to be an important aspect of programming, computing education research on code reading receives little attention [4]. We have limited knowledge about the reading and comprehension process of students and we do not know much about code reading patterns of students while performing static code reading such as identifying a bug in a given program code. Having insights about this process would allow us to develop learning tasks that could probably help improve code reading and code comprehension of students. Thus, this study aims to analyze eye tracking data of students while searching for bugs in a program to determine their focus of attention and identify common code reading patterns of students of high performing and low performing students.

1.1 Research Questions

We attempt to answer the following questions:

1. Is there a difference in the visual effort of high performing and low performing students in the task of finding bugs?
2. How can we identify the common code reading patterns that students of different skill levels use?

2.RELATED LITERATURE

2.1.Eye Tracking Studies on Visual Effort in Code Reading

Eye movements are closely linked to the allocation of attention and the most commonly studied aspect of eye movement behavior are saccades and fixations. Visual information processing occurs during fixations and that no such processing occurs during sac-

cades. The study of [21] used fixation durations and fixations counts on the defined AOIS of the source code to measure the visual effort of novices and non-novices with a task of finding bugs in C++ and Python. Results revealed that novices had lower fixation counts on all the buggy lines of code compared to non-novices. However, novices had higher fixation durations on all buggy lines of code in Python while non-novices had mostly same fixation durations in both C++ and Python. In contrast with the findings of [21], a recent study [7] which aims to analyze and interpret the gaze behavior of participants while detecting bugs in C revealed that experts focused more on the buggy lines of code whereas novices scanned the complete code which indicates more searching to detect bugs. While it has been shown that no visual information processing that occurs during saccades, the study of [19] used fixations and saccades in examining the visual processes of novice and advanced programmers who were asked to identify an error or provide the output of the code. The result of study showed that advanced programmers had shorter fixations and saccades which suggests that they can easily understand the code compared to novices.

The studies presented information about the difference in the visual effort of novices and non-novices while performing tasks such as bug finding and providing output of the code. The visual effort can be measured using fixations wherein visual processing occurs. This study also use fixation counts and fixation durations to measure the visual effort of high performing and low performing students while finding bugs in the source code.

2.2.Eye Tracking Scanpath Analysis Techniques

There are several algorithms and visualization techniques that have been introduced to analyze eye tracking scanpaths for different purposes. One of the most widely used algorithm to compare two scanpaths which are represented in terms of Areas of Interests (AOIs) is the Levenshtein Distance or String-edit algorithm [8]. This algorithm calculates a distance between two scanpaths by transforming one scanpath to another using minimum number of operations which are insertion, deletion and substitution. For example, the distance between ABCD and ABCE is calculated as one because it is sufficient to substitute D with E or vice versa. Though this algorithm can be used to compare the similarity of scanpaths, the algorithm cannot discover a common scanpath for multiple scanpaths.

eMINE Scanpath Algorithm [9] was developed to address the problem of being reductionist. This algorithm is composed of the String-edit algorithm and Longest Common Subsequence (LCS) technique. This algorithm employs first the String-edit algorithm by choosing the two most similar scanpaths. The LCS is then applied to these two scanpaths to find their common scanpath. The chosen scanpaths are removed from the list of scanpaths and then their common scanpath is added to the list. This process is repeated until there is a single scanpath left in the list. The single scanpath is then abstracted to provide the common scanpath. eMINE attempts to address the reductionist problem of the Dotplots algorithm [13] by using the String-edit algorithm and the LCS technique together. However, it still uses hierarchical clustering which means some visual elements can be lost at the intermediate levels. The algorithm is likely to produce very short common scanpaths which are not useful for further processing.

Scanpath Trending Analysis algorithm [10, 11] uses a multi-pass approach. It is consists of three stages: (1) Preliminary Stage, (2) First Pass, and (3) Second Pass. The preliminary stage correlates fixations with the visual elements or the specified Areas of Interests (AOIs). As a result, the individual scanpaths are generated and are represented in terms of the visual elements. The next stage which is the first pass is responsible for analyzing the individual scanpaths to identify the trending visual elements by considering the total number of fixations and the total duration of fixations on the visual elements. When the trending visual element instances are identified, other visual element instances that were not identified as trending visual element instances are removed from the individual scanpaths. The last stage in the algorithm is the second pass wherein the trending scanpath is constructed. The STA algorithm abstracts the individual scanpaths and then calculates the priority value (ψ) for each visual element instance in each individual scanpath with Equation 1 where P is the visual element instance position in the individual scanpath (the first position is zero) and L is the length of the individual scanpath. We take the max and min values as 1 and 0.1 respectively wherein the first visual element instance is given 1 point and 0.1 point is given to the last visual element instance. When all of the priority values are calculated, the total priority value (Ψ) for each instance is calculated with Equation 2 where n is the number of individual scanpaths. The algorithm then positions the instances into the trending scanpath based on their total priority values from highest priority to lowest. If multiple instances have the same total priority value, their total duration and their total number of fixations on the instances are compared to determine their position in the trending scanpath. Once the trending visual element instances are positioned in the trending scanpath, their numbers are eliminated (e.g. E1 → E) and then the consecutive repetitions are removed (e.g. EBCEE → EBCE). Thus, the trending scanpath is represented in terms of the visual elements.

$$\psi = 1 - P \cdot \frac{\max - \min}{L - 1} \quad (1) \qquad \Psi = \sum_{i=1}^{n} \psi_i \quad (2)$$

The reviewed scanpath analysis techniques provided information on the advantages and disadvantages of applying these algorithms to analyze a common scanpath of multiple scanpaths. String-edit algorithm can be used to compute the similarity of two scanpaths but this algorithm could not be used to discover a common scanpath for multiple scanpaths. eMINE and STA on the other hand, are intended for identifying common scanpaths on web pages. STA algorithm has been applied to eye tracking data to determine the common scanpath followed by the web users while browsing and searching information in a web page. It has been evaluated and results show that the algorithm was able to generate a common scanpath that include the visual elements in the required sequence for the successful completion of the browsing and search task as well as the visual elements shared by all individual scanpaths. Compared to other Common Scanpath Identification algorithms (eMINE, eyePatterns, Dotplots, SPAM), STA yielded a trending scanpath with greater similarity to the individual scanpaths [10]. The results of the evaluation of STA in generating common scanpath followed by web users while browsing and

searching information in the web and the similarity of the information search task and bug finding task provided information that STA can also be applied to determine the common code reading patterns of students of different skill levels while finding bugs in a program.

2.3.Eye Tracking Studies on Code Reading Patterns

Source code comprehension is considered as a special type of reading task [3] and research on reading natural language has provided insights about reading patterns. The study of Busjahn, et al. [5] investigated the reading patterns of novices and experts while reading natural language texts and source code. Results showed that novices read source code in a less linear manner than natural language texts and experts read source codes less linearly than novices. The findings of [5] is similar to the findings of [15] that reading code is far from the conventional mostly-linear order employed when reading natural language texts. Reading code exhibits sequence of patterns such as scanning, jumping ahead to look for ideas, jumping back to verify details and other scanpath events identified by the study. However, the findings of this study should be explored further before a general picture of code reading will emerge since the patterns employed by the subjects and their order are highly individualistic.

The study of Lin et al. [17] conducted a sequential analysis of the path of student's gaze to reveal significant sequences of areas examined. These significant gaze path sequences were then compared to those students with different debugging performance. Findings of study revealed that high performance students traced programs in a more logical manner, whereas low-performance students tended to stick to a line by line sequence and experience difficulty in understanding the program's logic. In addition, low-performing students often jump directly to certain statement to find bugs, without following the program's logic.

The differences between the reading patterns of novices and experts were examined by Busjahn, et al. [5] and Lin et al. [17] while the study of Jbara and Feitelson [15] describes the scanpaths of all the subjects. While it is evident in their findings that code reading follows a non-linear pattern, we still have limited knowledge on what visual strategy high performing and low performing students use while finding bugs in the source code. The study of [15] was able to identify scanpath events from the individual scanpaths but was not able to generalize the findings, thus further exploration is needed to analyze the scanpaths of students to provide a general code reading pattern or visual strategy. This study attempts to determine the common code reading patterns of high performing and low performing students using the STA algorithm.

3.METHODOLOGY

This paper is an analysis of a larger eye tracking study on programmer tracing and debugging skills as well as development of higher education's capacity to conduct eye tracking research. The methods discussed here are also discussed in [22] which characterizes the extent of collaboration of pairs of novice programmers based on prior knowledge while tracing and debugging code fragments using cross-recurrence quantification analysis (CQRA), [23] which focuses on the assessment of who among the pairs of programmers collaborated the most regardless of expertise, and [24] which focus on the comparison of the collaboration of pairs of programmers based on prior knowledge and acquaintanceship while tracing and debugging code fragments.

3.1.Participants

Eye movement data were collected from 16 students of Ateneo de Davao University. There were 12 short Java programs with defects that the participants viewed in one setting. Out of the 12 short programs, 9 of them had 3 defects while 3 had 1 defect each. We identified two types of participant groupings based on their total scores on finding bugs. The first group consists of 8 participants with High Scores and 8 participants with Low scores. This was done to identify the common scanpath followed by high performing and low performing students.

3.2.Experimental Setup and Procedure

We used the Gazepoint eye tracker in this study. The device operates a sampling rate of 60Hz and an accuracy of 0.5-1 degree. The screen resolution was set to 1024x768 and the source code was presented in full screen window. Figure 1 shows the standard setup of the Eye Tracking experiment.

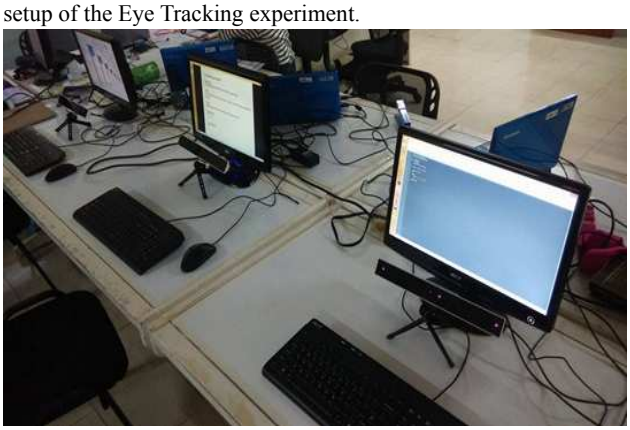

Figure 1. Standard Set-up of the Eye Tracking Experiment

Participants were asked to view 12 programs with defects in one setting. For this study, we selected one program code with 3 defects out of the 12 programs. The selected program code as stimulus is a code that allows the user to display all Q prime numbers in the set of Java programs. The program contains conditional statements and repetition structures. We defined the AOIs consisting of the lines of code of the program using the OGAMA tool to identify the coordinates of the AOIs. The AOIs were defined per line of codes to determine which among these lines of codes are frequently fixated by the participants and identify the linearity of source code reading employed by the participants. Figure 2 shows the AOIs of the program code. The AOIs are labeled based according to their line numbers but were later converted to the letters of the alphabet to be able to apply the String-edit algorithm for the similarity/dissimilarity calculation of the individual scanpaths and the trending scanpath.

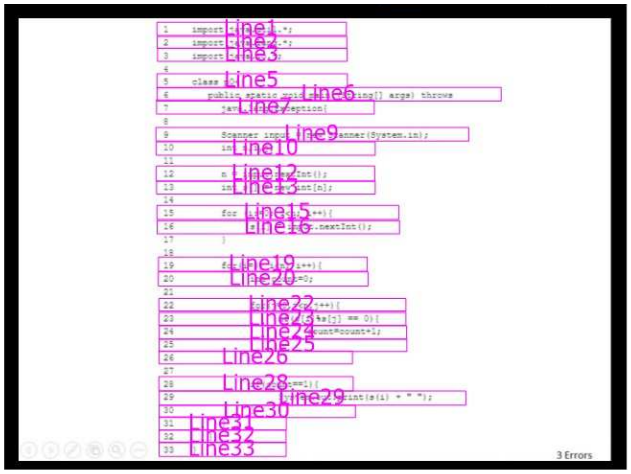

Figure 2. AOIs of the program code

3.3.Data Analysis

This subsection discusses the data analysis that was done to answer the research questions stated in Section 1.1. The Q Prime program is consists of 33 lines of code and 25 AOIs were defined. This program has three syntax errors located on lines 15, 22 and 29. Line 15 had an extra colon character in the initialization part of the first for loop (i:=0;), line 22 is also located in the for loop specifically the second loop which replaces the semicolon with comma in the initialization and condition part. The last error located on line 29 replaces the bracket sign with a parenthesis sign to print the contents of the array. To determine the difference in the visual effort of high performing and low performing students, fixation duration and number of fixations are used. The total number of eye fixations on the entire stimulus (TotalFC), total time (ms) of all fixations on the entire stimulus (TotalFD), total number of eye fixations on the buggy lines of code (FCBugLine), and total time (ms) of all fixations on the buggy lines of code were computed to measure the visual effort exerted by high performing and low performing students.

To answer research question 2 stated in Section 1.1, Scanpath Trend Analysis algorithm [11] was employed to identify the common code reading patterns of students while searching for bugs in a program code. The algorithm has three stages namely: (1) Preliminary stage, (2) First pass, and (3) Second pass. The preliminary stage relates the eye tracking data with the visual elements (AOIs) to prepare the individual scanpaths. The output of this stage would be the individual scanpaths of the students while searching for bugs. Table 1 illustrates the sample individual scanpaths of the students with their corresponding fixation durations enclosed in brackets. Note that the data in the table were not completely listed.

Table 1. Sample Individual Scanpaths of High Performing Students

ID	Individual Scanpath
1	T [538] S[197] X [191] Y [206.05] T [315] L [624] A [995.36] D [483.52] B[262.57]
2	J [199.71] J [279.3] W [321.29] Y [263.67] Z [332] I [322] H [516.11] K [231.93] J [233.89] G [305.18]
3	T[232.42] T[138.67] J[233.4] D[276.37] K[186.52] Z[326.17] W[432.62] T[443.36] Q[185.55] D[280]
4	G[410.16] F[212.89] H[82.03] A[361.33] H[197.27] B[425.78] B[212.89] G[330.08] D[361.33] D[343.75] E[722.66] F[246.09] A[279.3] B[246.09] D[363.28] A[226.56] E[328.13] E[228.52] G[312.5] G[558.59]
5	M[570.31] M[653.32] M[801.76] L[209.96] L[422.85] L[263.67] L[345.7] M[207.03] X[178.91] V[483.4] O[455.08] O[1037.11] P[991.21] P[480.47] O[249.02] O[416.99] P[177.15]

Once the individual scanpaths are prepared, the trending visual elements were identified using their occurrence frequencies (i.e. the number of fixations on a visual element) and their fixation durations. Analyses of the individual scanpaths to determine the threshold values for the occurrence frequencies and durations were performed to decide whether the visual element is trending or not. As can be seen from the individual scanpath with an ID 5 in Table 1, visual elements can be repeated which can be consecutive repetition such as MMML or non-consecutive repetition, such as MXM. Consecutive repetition occurs when a user fixates the same visual element more than one time in succession whereas the non-consecutive repetition occurs when a user fixates a visual element (M) and fixates another visual element (MX) and then goes back to the previous visual element (MXM). Each visit of a particular visual element is referred to as a visual element instance. In the individual scanpath with an ID 5 in Table 1, the visual element M is visited two times (…..MMM…….M…..). The first instance is identified when the student fixated the visual element M three times in succession and the second instance is when the user fixated once in the visual element M before moving to the next visual element. Therefore, there are two instances of the visual element M in the individual scanpath with an ID 5.

After identifying the visual element instances, we applied the First Pass stage discussed in Section 2.2. The algorithm differentiates the instances by assigning them different numbers based on their total durations as the durations are correlated with information processing. Specifically, the first number is assigned to the visual element instance with the highest fixation duration of a particular visual element in an individual scanpath. For example, the first instance of the visual element M in the individual scanpath shown in Example 3.4.1 is numbered with one because its duration (M1

=507.31 + 653.32 + 801.76 = 1962.39 ms) is longer than another instance of the visual element M (M2 = 207.03 ms).

Example 3.4.1 An example of an individual scanpath in terms of visual element instances with the fixation durations
M1[570.31] M1[653.32] M1[801.76] L1[209.96] L1[422.85] L1[263.67] L1[345.7] M2[207.03] X1[178.91] V1[483.4] O1[455.08] O1[1037.11] P1[991.21] P1[480.47] O2[249.02] O2[416.99] P2[177.15]

Once the visual element instances are differentiated, the trending visual element instances were identified. When a particular visual element is fixated by all participants, this means that the visual element has important potential for processing. Thus, when the visual element instance is shared by all the individual scanpaths, it is considered as a trending visual element instance. The visual element instance which is not shared by all the individual scanpaths but receives the same attention as the shared instances were also considered as trending visual element instance. In order to identify these trending instances, we find the instances that are shared by all individual scanpaths and then computed the minimum total occurrence frequency and minimum total durations. When an instance has the same or higher total occurrence frequency than the minimum total occurrence frequency and the same or longer total duration than the minimum total duration of the shared instances of the individual scanpath, the instance is defined as a trending visual element. The visual element instances that do not satisfy the criteria were removed from the individual scanpaths. Table 2 illustrates the trending visual element instances of the individual scanpaths which are shown in Table 1.

Table 2. Sample Individual Scanpaths with the Trending Visual Element Instances after the First Pass

ID	Individual Scanpath
1	T1 [538.21] S1[197.08] X1 [191.35] Y1[206.05] T2[315] L1 [624] A1 [995.36] D1 [483.52] B1 [262.57]
2	J2 [199.71] J1 [279.3] W1 [321.29] Y1 [263.67] Z1 [332] I1 [322] H1 [516.11] K1[231.93] B1 [233.89] G1 [305.18]
3	T2[232.42] T3[138.67] J1[233.4] K1[186.52] Z1[326.17] W1[432.62] T1[443.36] Q1[185.55] D1[280]
4	G2[410.16] F2[212.89] H2[82.03] A1[361.33] H1[197.27] B1[425.78] B1[212.89] G3[330.08] D3[343.75] F1[246.09] B2[246.09] D1[363.28] G1[312.5] G1[558.59]
5	M1[570.31] M1[653.32] M1[801.76] L1[209.96] L1[422.85] L1[263.67] L1[345.7] M2[207.03] V1[483.4] O1[455.08] O1[1037.11] P1[991.21] P1[480.47] O2[249.02] O2[416.99]

Assuming that P2 and X1 in the individual scanpath are not defined as trending visual element instances, they were removed in the individual scanpath as shown in Table 2.
The next step was to identify the trending scanpath. A trending scanpath is constructed by clustering the individual scanpaths with the trending visual element instances in the individual scanpaths identified in the first pass. These visual element instances were located in the trending scanpath based on their overall positions in the individual scanpaths. The individual scanpaths are abstracted by combining the same visual element instances. Example 3.4.2 shows the abstracted version of the individual scanpath with an ID 5 in Table 2. The total number of occurrences and the total duration for each visual element instance were also saved for future reference.

Example 3.4.2. An example of the abstracted individual scanpath in terms of the trending visual element instances
M1 L1 M2 V1 O1 P1 O2

To find out the proper positions of the trending visual element instances in the trending scanpath, the priority value (ψ) for each visual element instance for each individual scanpath are computed with the formula shown in Equation 1, where P_i is the visual element instance position in the individual scanpath (the first position is zero) and L is the length of the individual scanpath. We take the max and min values as 1 and 0.1 respectively, to give 1 point to the first visual element instance and 0.1 points to the last visual element instance. Table 3 shows the priority values (ψ_i) of the visual element instances in the individual scanpath shown in Example 3.4.2.

Table 3. The priority values (ψi) of the visual element instances in the individual scanpath shown in Example 3.4.2.

Visual Element Instance	Priority Value (ψ_i) where z = 1-0.1/(7-1)
M1	1 – 0 * z = 1
L1	1 – 1 * z = 0.85
M2	1 – 2 * z = 0.70
V1	1 – 3 * z = 0.55
O1	1 – 4 * z = 0.4
P1	1 – 5 * z = 0.25
O2	1 – 6 * z = 0.1

Once the priority values are calculated, the next step in the algorithm was applied by calculating the total priority value (Ψ) for each visual element instance in all individual scanpaths using the formula shown in Equation 2 in Section 2.2. The total priority values of the visual element instances were ranked from highest to lowest to identify their position in the trending scanpath. The visual element instance with the highest total priority value is located at the beginning of the trending scanpath. When a trending visual element instance is not shared by all individual scanpaths

and has lower total priority value than the minimum total priority value of the shared visual element instances, it will not be included in the trending scanpath. In case more than one visual element instance have the same total priority value, the total duration and total number of occurrences will be used as reference to locate them into the trending scanpath. Once the trending scanpath is constructed, the numbers of the visual element
instances are removed. An example of a trending scanpath is MLMVOPO.

4.RESULTS AND DISCUSSION

Eye tracking tools collect eye movement data which can be analyzed to provide insights into a subject's focus of attention [14]. The eye tracking data collected from the 16 students of Ateneo de Davao University allowed us to determine the visual effort of students using the number of fixations on the AOIs and the number of fixation durations and determine the code reading patterns of students while performing bug finding task using the STA algorithm. Out of the 16 eye tracking data, 8 are high performing students and 8 are low-performing students. One eye tracking data from the high performing students was excluded in the analysis since only 16 fixations were correlated to the AOIs and a total time of 1,802.73 milliseconds was spent on these fixations.

4.1.Difference in the Visual Effort of High Performing and Low Performing Students

To answer research question 1, the total number of eye fixations on the entire stimulus (TotalFC), total time (ms) of all fixations on the entire stimulus (TotalFD), total number of eye fixations on the buggy lines of code (FCBugLine), and total time (ms) of all fixation durations on the buggy lines of code (FDBugLine) were computed to measure the visual effort exerted by high performing and low performing students. Tables 4 and 5 show the result of the calculation of the total number of fixations on the entire stimulus and total number of eye fixations on the buggy lines of code of both high and low performing students.

Table 4. Total Number of Eye Fixations on the Entire Stimulus

Low Performing ID	TotalFC	High Performing ID	TotalFC
S1	183	S5	257
S2	210	S7	168
S3	390	S8	163
S4	427	S10	227
S9	175	S11	232
S12	137	S13	125
S14	502	S15	135
S16	774		
Total	**2,798**	Total	**1,307**
Mean	**350**	Mean	**187**

Table 4 shows that the total number of fixations on the entire stimulus (TotalFC) of low performing students is 2,798 which is higher compared to 1,307 of high performing students. Even if we add the total number of fixations of S6, the result would still show that low performing students have more fixation counts.

Table 5. Total Number of Eye Fixations on the Buggy Lines of Code

Low Performing ID	FCBugLine			High Performing ID	FCBugLine		
	15	**22**	**29**		**15**	**22**	**29**
S1	8	13	2	S5	17	21	1
S2	14	15	7	S7	7	15	2
S3	14	30	0	S8	15	10	3
S4	9	7	2	S10	22	12	3
S9	12	10	2	S11	3	8	3
S12	13	9	3	S13	0	2	2
S14	12	44	13	S15	18	12	2
S16	22	37	50				
Total	**104**	**165**	**79**	Total	**82**	**80**	**16**
Mean	**13**	**21**	**10**	Mean	**12**	**11**	**2**

Table 5 shows that low performing students have higher fixation counts on the buggy lines of code compared to high performing students. The result is in contrast with the findings of the study of [21] that novices have lower fixation counts on all buggy lines of code than non-novices. Based on the fixation counts of the buggy lines of code of low performing students, line number 15 ranked 8 out of the 25 lines, line 22 received the second highest number of fixations, and line 29 ranked 11. Though line 22 has the second highest number of fixations, 3 out of the 8 students were not able to correctly identify the bug. Lines 15 and 22 had a ranking of 3 and 4 respectively, based on the number of fixations of high performing students while line 29 had a rank of 15. Out of the 7 students, all of the errors were correctly identified however, S13 and S15 received a score of 0.5 since the mark made was just close to the error in line 29. This means that even if low performing students looked at the buggy lines of code more times than the high performing students, they are still unable to correctly identify the error.

To determine the time spent by high performing and low performing students on the entire stimulus and the buggy lines of code, the total fixation durations was computed. Table 3 and Table 4 present the result of the computation.

Table 6. Total Time of all Fixations on the Entire Stimulus

Low Performing ID	TotalFD (ms)	High Performing ID	TotalFD (ms)
S1	50,656.94	S5	56,457.08
S2	69,451.17	S7	48,964.47
S3	12,5301.24	S8	72,266.58
S4	13,6338.85	S10	97,092.8
S9	62,861.83	S11	71,710.93
S12	37,935.09	S13	42,806.84
S14	10,1074.66	S15	73,746.96
S16	23,4255.45		
Total	**81,7875.23**	Total	**46,3045.66**
Mean	**102,234.4**	Mean	**66,149.38**

Table 6 shows the total time spent by the high performing and low performing students while identifying the errors of the short program. Result showed that the difference on the total time spent by low performing students is more than 57% higher compared to the total time spent by high performing students in finding bugs on the source code.

Table 7 shows the total time of all fixations by the students on the buggy lines of code. Based from the table, low performing students had higher fixation durations on lines 22 and 29 while they spent shorter fixation durations on line 15 compared to high performing students. The finding shows that it is aligned to the result of [21] and [19] that advanced programmers had shorter fixations and saccades which suggests that they can easily understand the program compared to novices. However, the study of [7] revealed that experts focused more on buggy lines of code compared to novices which is contrary to the findings of [21] and [19].

The findings in the analysis of visual effort suggest that higher fixation durations and larger number of fixations indicate difficulty in identifying the buggy lines of code. High performing students spent less time and smaller number of fixations compared to low performing students because they can easily comprehend the program thus, being able to correctly identify the bugs. Since there is only limited sample data for this analysis, a larger sample data would be analyzed to get a general picture of the visual effort spent by high performing and low performing students. This preliminary analysis has provided insights on the visual attention of students while performing static code reading type of programming task with the focus of finding bugs in the program.

4.2.Common Code Reading Patterns of High Performing and Low Performing Students

The eye tracking data was correlated to the AOIs to construct an individual scanpath sequence. STA algorithm was used to analyze the individual scanpath sequence by clustering the fixation points to generate a trending scanpath or a common scanpath. This scanpath could be used to describe the code reading patterns used by students of high performing and low performing students while performing bug finding task. Table 8 and Table 9 show the sequence of individual scanpaths after identifying the trending visual elements. Trending visual elements are those AOIs which are shared by all participants and have received the same or higher amount of attention from the participants.

Table 7. Total Time of all Fixations on the Buggy Lines of Code

Low Performing ID	FDBugLine			High Performing ID	FDBugLine		
	15	**22**	**29**		**15**	**22**	**29**
S1	3,220.39	13,214.67	0	S5	8,910.18	8,365.24	2,78.32
S2	6,971.2	65,20.02	1,099.61	S7	3,728.51	8,937.51	929.68
S3	7,606.2	16,622.59	4,020.26	S8	12,519.53	7,447.27	1,338.87
S4	3,691.41	3,307.61	6,354.49	S10	15,892.59	7,206.5	1,326.17
S9	11,660.39	9,089.48	2,637.21	S11	1,443.35	2,976.56	2,917.48
S12	5,654.8	5,000.48	767.83	S13	0	1,438.17	247.35
S14	41,42.83	21,055.67	1,320.06	S15	21,034.92	19,138.42	1,279.2
S16	12,145.0	23,710.93	3,020.51				
Total	**55,092.22**	**98,521.45**	**19,219.97**	**Total**	**63,529.08**	**55,509.67**	**8,317.07**
Mean	**6,886.53**	**12,315.18**	**2,402.5**	**Mean**	**9,075.58**	**7,929.95**	**1,188.15**

Table 8. Scanpath Lengths of High Performing Students

ID	Length of Scanpath
S5	26
S7	26
S8	24
S10	28
S11	25
S13	26
S15	22
Average Scanpath Length	**25.5**

Table 8 shows the scanpath lengths of high performing students while finding bugs in the program. The average scanpath length is 25.5 while the scanpath length of the common scanpath is 31. The length of the common scan path is slightly longer compared to the length of the individual scanpaths.

Table 9. Scanpath Lengths of Low Performing Students

ID	Length of Scanpath
S1	50
S2	46
S3	54
S4	57
S9	46
S12	42
S14	53
S16	52
Average Scanpath Length	52.33

Table 9 shows the length of individual scanpaths of low performing students with an average scanpath length of 52.33 while finding bugs in the program. Compared to the average length of individual scanpaths length of high performing students, the average scanpath length of students with low scores is longer.

The individual scanpaths were used in generating the common scanpath of students by calculating the total priority values of the visual element instances. After applying the Second pass of the STA algorithm described in Section 3.3, the common scanpaths of students were identified.

Table 10 shows the length of scanpath of high and low performing students. The scanpath length of high performing students is 31 which is smaller compared to the scanpath length of low performing students which is 58. This means that low performing students have more fixations compared to high performing students.

Table 10. Common Scanpath Length of High and Low Performing Students

Student Group	Length of Scanpath
High Performing	31
Low Performing	58

Since the STA algorithm constructs a common scanpath based on individual scanpaths, the common scanpath should be similar to the individual scanpaths. It is not possible for the common scanpath to have 100% similarity to the individual scanpaths unless all the individual scanpaths are the same. Therefore, the similarities between the common scanpath and the individual scanpaths are strongly related to the similarities between the individual scanpaths. It is expected that the median similarity of the trending scanpath to the individual scanpaths is equal or greater than the minimum similarity between the individual scanpaths. Using the String-edit algorithm [9], the similarity of the individual scanpaths and the common scanpath's similarity with the individual scanpaths were computed. Table 11 shows the result of the calculation of the similarity of the individual scanpaths with the common scanpath of high and low performing students.

Table 11. Similarity of Common Scanpath with the Individual Scanpaths of High and Low Performing Students

Comparisons (Low Performing)	Similarity	Comparisons (High Performing)	Similarity
S1-Common Scanpath	18.97	S5-Common Scanpath	25.81
S2-Common Scanpath	24.14	S7-Common Scanpath	25.81
S3-Common Scanpath	22.41	S8-Common Scanpath	25.81
S4-Common Scanpath	15.52	S10-Common Scanpath	22.58
S9-Common Scanpath	25.86	S11-Common Scanpath	29.03
S12-Common Scanpath	27.59	S13-Common Scanpath	22.58
S14-Common Scanpath	27.59	S15-Common Scanpath	25.81
S16-Common Scanpath	25.86		
Median	**25**	**Median**	**25.81**

Table 11 shows that the median similarity of the common scanpath with the individual scanpaths of low performing students is 25, while the median similarity of high performing students is 25.81. We can see from the table that the similarity of the common scanpath with the individual scanpaths of low and high per-

forming students is almost similar. To determine if STA was able to generate a common scanpath that is similar to the individual scanpath, Table 12 shows the similarity scores between the individual scanpaths of both skill levels. It should be expected that the median similarity of the common scanpath with the individual scanpath is equal or higher than the similarity score of the individual scanpaths.

Table 12. Similarity between the Individual Scanpaths of High and Low Performing Students

Comparisons (Low Performing)	Similarity	Comparisons (High Performing)	Similarity
S1-S2	24.00	S5-S7	30.77
S1-S3	20.37	S5-S8	34.62
S1-S4	29.82	S5-S10	28.57
S1-S9	28.00	S5-S11	34.62
S1-S12	28.00	S5-S13	42.31
S1-S14	22.64	S5-S15	26.92
S1-S16	17.31	S7-S8	46.15
S2-S3	22.22	S7-S10	32.14
S2-S4	26.32	S7-S11	42.31
S2-S9	23.91	S7-S13	34.62
S2-S12	30.43	S7-S15	30.77
S2-S14	35.85	S8-S10	39.29
S2-S16	21.15	S8-S11	28.00
S3-S4	21.05	S8-S13	30.77
S3-S9	25.93	S8-S15	37.5
S3-S12	25.93	S10-S11	32.14
S3-S14	22.22	S10-S13	17.86
S3-S16	20.37	S10-S15	28.57
S4-S9	26.32	S11-S13	42.31
S4-S12	21.05	S11-S15	12.00
S4-S14	17.54	S13-S15	23.08
S4-S16	17.54		
S9-S12	28.26		
S9-S14	32.08		
S9-S16	28.85		
S12-S14	26.42		
S12-S16	21.15		
S14-S16	16.98		
Median	**23.96**	**Median**	**32.14**

Table 12 shows that the median similarity score of the individual scanpaths of low performing students is 23.96 which is lower than the median similarity score of the common scanpath with the individual scanpaths which is 25. The median similarity score of the individual scanpaths of high performing students is 32.14 which is higher than the median similarity score of the common scanpath with the individual scanpaths which is 25.81. The result shows that the STA algorithm was able to generate a common scanpath which is similar to the individual scanpaths although the similarity score of the individual scanpaths of high performing students is higher compared to the similarity score of the common scanpath. STA algorithm performs better in identifying the common scanpath followed by low performing students. It may be because high performing students have individual strategies in reading code while low performing students apply similar strategies.

To determine the code reading patterns between the high and low performing students, a graph was generated to show the sequence of eye movements. Figure 3 and Figure 4 show the common scanpaths of high and low performing students.

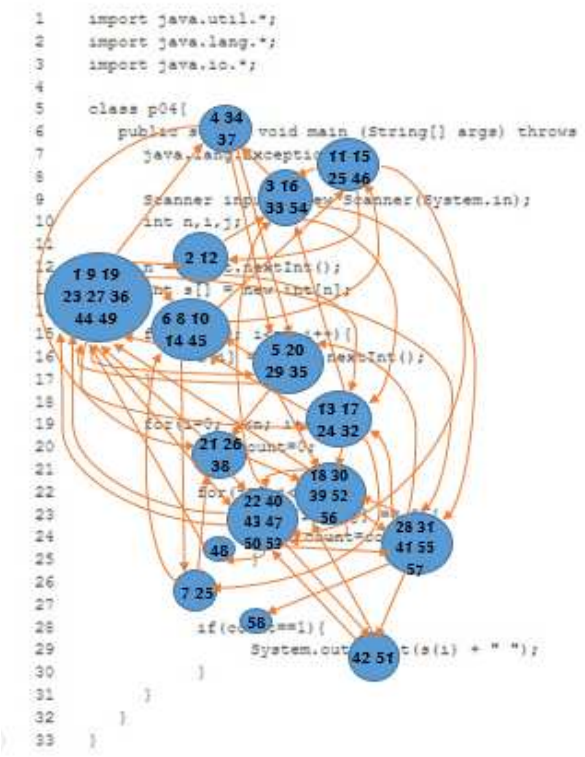

Figure 3. Common Code Reading Patterns of Low Performing Students

The size of the blue circles represents the attention given for the line number and the numbers inside represents the common scanpath sequence of gazes for that line. The directional arrows represents the sequence origin and destination.

Figure 3 shows the sequence of gazes of the low performing students and we can see that their common scanpath starts reading from line 13 followed by look back movements then a forward

jump with a series of regressions. The code reading patterns of low performing students jump through the different lines of code in a non-linear fashion. This is in accordance with the findings of the study of [17] that low-performing students often jump directly to certain statement to find bugs, without following the program's logic.

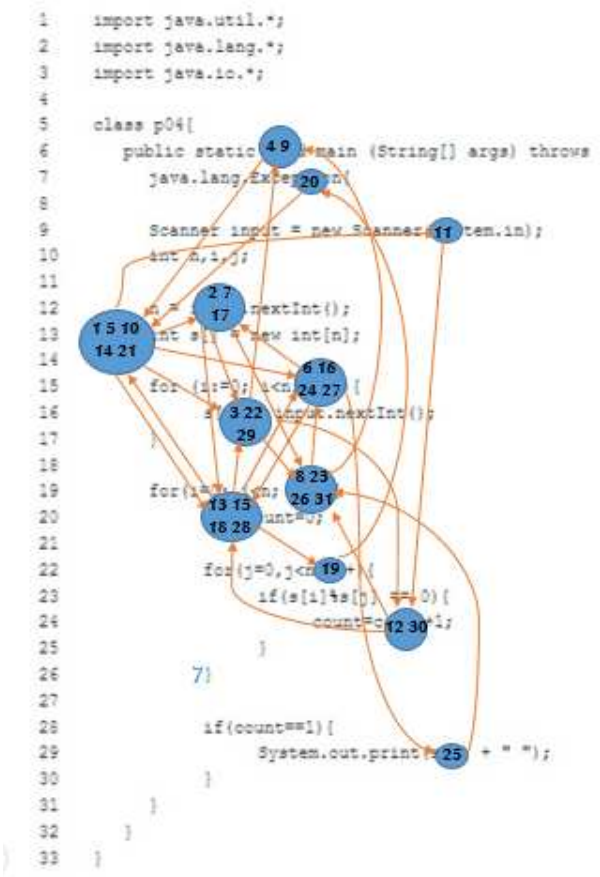

Figure 4. Common Code Reading Patterns of High Performing Students

Based from the figure above, we can see that high performing students starts reading from line 13 similar to low performing students and continue to read the code in a non-linear manner and is characterized by longer saccades compared to the scanpath of low performing students. We can also see that there are minimal regressions made by high performing students compared to low performing students. According to the findings of the study of [17] experts traced programs in a more logical manner and less linearly than low performing students or novices [5, 17]. We can also see that the high performing students were able to scan through the entire source code from line 7 which is the line where main function is located to line 29 which is the last statement of the code and where the third error is located. The pattern also shows that the eye movements passed through the errors in the code which are lines 15, 22 and 29. According to [7] experts perceive the program better and hence able to fixate on the error regions. The generated common scanpath exhibits the characteristics of high performing students.

5.CONCLUSION AND FUTURE WORKS

The study aimed to determine the visual effort spent by low performing and high performing students while finding bugs in the program. Results show that the number of fixations and fixation durations show how much effort is exerted to be able to find the bugs in the program. The larger the number of fixations and the more time spent in the buggy lines of code indicates difficulty in correctly identifying the bugs in the program.

The result of the study of [15] mentioned that no general picture of code reading can be concluded since the patterns employed by the subjects and their order are highly individualistic. With the use of STA algorithm, we were able to generate a common scanpath which could be used to generalize the code reading patterns of the students with different skill level. The generated common scanpath exhibits the reading patterns that were observed by previous research that high performing students read code in a more logical manner and that they have longer saccades which means that they read code in a non-linear fashion. High performing students perceive the program better thus, were able to fixate on the error regions and correctly identified the bugs. Low performing students on the other hand, often jump directly to certain statement to find bugs, without following the program's logic and is characterized by frequent regressions. By exploring the strategies employed by high performing students, we could develop teaching materials that could help low performing students in improving their source code reading skill. These strategies used by high performing students for carrying out the coordination and integration of goals and plans that underlie program code could be explicitly taught. Further, teaching students to consciously employ code reading strategies used by high performing students would help them develop their own effective approach.

The result of the study could contribute to the evidences that STA algorithm could be used to identify common scanpath of multiple scanpaths. However, the initial analysis that was done in this study is just a preliminary investigation of the potential effectiveness of the STA algorithm in identifying a common scanpath because of the small data sample. Further studies involving larger data sample would be conducted to be able to validate and generalize the result of this study.

6.ACKNOWLEDGEMENT

We thank the Ateneo de Naga, Ateneo de Manila, Ateneo de Davao, University of Southeastern Philippines, Private Education Assistance Committee of the Fund for Assistance to Private Education for the grant entitled "Analysis of Novice Programmer Tracing and Debugging Skills using Eye Tracking Data" and Ateneo de Manila University's Loyola Schools Scholarly Work Faculty grant entitled "Building Higher Education's Capacity to Conduct Eye-tracking Research using the Analysis of Novice Programmer Tracing and Debugging Skills as a Proof of Concept".

7.REFERENCES

1 Aschwanden, C & Crosby., M. E. 2005. Code Scanning Patterns in Program Comprehension.

2 Bednarik, R., Myller, N., Sutinen, E., and Tukiainen, M. 2006. Program Visualization: Comparing Eye-Tracking Patterns with Comprehension Summaries and Performance. In *Proceedings of the 18th Workshop of the Psychology of Programming Interest Group.* University of Sussex, September 2006. pp. 68-82

3 Bednarik, R., Busjahn, T., Schulte, C., and Tamm, S. (Eds.). 2016. Eye Movements in Programming Models to Data: *Proceedings of the Third International Workshop,* University of Eastern Finland, Joensuu Finland, 25-26.

4 Busjahn, T. and Schulte, C. 2013. The use of code reading in teaching programming. In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research* (Koli Calling '13). ACM, New York, NY, USA, 3-11. DOI=_http://dx.doi.org/10.1145/2526968.2526969

5 Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J.H., Schulte, C., Sharif, B., and Tamm, S.. 2015. 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, Piscataway, NJ, USA, 255-265.

6 Busjahn, T., Schulte, C., and Busjahn, A. 2011. 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). ACM, New York, NY, USA, 1-9. DOI=http://dx.doi.org/10.1145/2094131.2094133

7 Chandrika. K. R., and Amudha, J. 2017. An Eye Tracking Study to Understand the Visual Perception Behavior while Source Code Comprehension. International Journal of Control Theory and Applications. International Science Press vol. 10(19)

8 Duchowski, A. T., Driver, J., Jolaoso, S., Tan, W., Ramey, B. N., and Robbins, A. 2010. Scanpath Comparison Revisited. In Proceedings of the 2010 Symposium on Eye Tracking Research and Applications. ACM, New York, NY, USA, 219-226.

9 Eraslan, S., Yesilada, Y., and Harper, S. 2014. Identifying Patterns in Eyetracking Scanpaths in Terms of Visual Elements of Web Pages. In Web Engineering, Sven Casteleyn, Gustavo Rossi, and Marco Winckler (Eds.). LNCS, Vol. 8541. Springer International Publishing, 163-180

10 Eraslan, S., Yesilada, Y., and Harper, S. 2016. Trends in Eye Tracking Scanpaths: Segmentation Effect?. In Proceedings of the 27th ACM Conference on Hypertext and Social Media(HT '16). ACM, New York, NY, USA, 15-25. DOI: https://doi.org/10.1145/2914586.2914591

11 Eraslan, S., Yesilada, Y., and Harper, S. 2016. Scanpath Trend Analysis on Web Pages: Clustering Eye Tracking Scanpaths. ACM Trans. Web 10, 4, Article 20 (November 2016), 35 pages. DOI: https://doi.org/10.1145/2970818

12 Fan, Q. 2010. “The effects of beacons, comments, and tasks on program comprehension process in software maintenance,” PhD thesis, University of Maryland at Baltimore County, Catonsville, MD, USA.

13 Goldberg, J. H., and Helfman, J. I. 2010. Scanpath Clustering and Aggregation. In Proceedings of the 2010 Symposium on Eye Tracking Research and Applications, ACM, New York, USA, 18-27.

14 Gueheneuc, Y. G., Kagdi, H., and Maletic, J. I. 2009. Working Session: Using eye-tracking to understand program comprehension. 2009 IEEE 17th International Conference on Program Comprehension, Vancouver, BC, pp. 278-279. DOI: 10.1109/ICPC.2009.5090057

15 Jbara, A., and Feitelson, D. G. 2007. "How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking," 2015 IEEE 23rd International Conference on Program Comprehension, Florence, pp. 244-254. doi: 10.1109/ICPC.2015.35

16 Jones, S.J., and Burnett, G. E. 2007. Spatial skills and navigation of source code. In Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education (ITiCSE '07). ACM, New York, NY, USA, 231-235. DOI: http://dx.doi.org/10.1145/1268784.1268852

17 Lin, Y. T., Wu, C. C., Hou, T. Y., Lin, Y. C., Yang, F. Y., & Chang, C. H. 2016. Tracking students’ cognitive processes during program debugging—An eye-movement approach. IEEE transactions on education, 59(3), 175-186.

18 Melo, J., Narcizo, F. B., Hansen, D. W., Brabrand, C., and Wasowski, A. 2017. "Variability through the Eyes of the Programmer," 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), Buenos Aires, 2017, pp. 34-44. doi: 10.1109/ICPC.2017.34

19 Nivala, M., Hauser, F.,Mottok, J., and Gruber, H. 2016. "Developing visual expertise in software engineering: An eye tracking study," 2016 IEEE Global Engineering Education Conference (EDUCON), Abu Dhabi, 2016, pp. 613-620. doi: 10.1109/EDUCON.2016.7474614

20 Schroter, I, Kruger, J., Seigmund, J., and Leich, T. 2017. Comprehending studies on program comprehension. In *Proceedings of the 25th International Conference on Program Comprehension (ICPC '17)*. IEEE Press, Piscataway, NJ, USA, 308-311. DOI: https://doi.org/10.1109/ICPC.2017.9

21 Turner, R., Falcone, M., Sharif, B., and Lazar, A. 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 (ETRA '14). ACM, New York, NY, USA, 231-234. DOI: https://doi.org/10.1145/2578153.2578218

22 Villamor, M. and Rodrigo, M. M. 2017. Charaterizing Collaboration in the Pair Program Tracing and Debugging Eye-Tracking Experiment: A Preliminary Analysis. In *Proceedings of the 10th International Conference on Educational Data Mining*. (pp. 174-179)

23 Villamor, M., Paredes, Y.V., Samaco, J.D., Cortez, J.F., Martinez, J., and Rodrigo, M.M.. 2017. Assessing the Collaboration and Quality in the Pair Program Tracing and Debugging Eye-Tracking Experiment. In: Andre E. Baker., Hu, X. Rodrigo, M, du Buolay B. (eds) Artificial Intelligence in Education. AIED 2017. Lecture Notes in Computer Science, vol 10331. Springer, Cham. DOI: https://doi.org/10.1007/978-3-319-614250-0_67

24 Villamor, M. and Rodrigo, M. M. 2017. Impact of Both Prior Knowledge and Acquaintanceship on Collaboration and Performance: A Pair Program Tracing and Debugging Eye-Tracking Experiment. In *Proceedings of the 25th International Conference on Computers in Education.* New Zealand: Asia Pacific Society for Computers in Education

25 Yenigalla, L.K. 2014. How Novices Read Source Code. Youngstown State University. Masters Thesis.