# Visual Attention Patterns in Finding Source Code Defects

CHRISTINE LOURRINE S. TABLATIN
Information Technology Department,
Pangasinan State University,
San Vicente, Urdaneta City, Pangasinan,
PHILIPPINES

*Abstract:* - Existing research used visual effort metrics to determine the visual attention patterns of participants with varying skill levels while finding source code defects. While most of the findings of these studies agree on the results for fixation count metrics, there are differences in the results for fixation duration metrics. Therefore, there is a need for further investigations on the use of visual effort metrics to determine the difference in the visual effort of experts and novices between multiple programs. Thus, we aimed to identify the factors affecting the varying results on fixation duration metrics and validate the results on fixation count metrics. We used visual effort metrics to identify the visual attention patterns of high and low-performing students engaged in defect-finding tasks on multiple programs. We performed statistical tests on the total fixation count, fixation counts on the error lines, total fixation duration, and fixation duration on the error lines to determine the difference in the visual attention patterns between the groups. Among the fixation metrics used, only the total fixation duration metric revealed a significant difference between the high and low-performing students across all programs. High-performing students spent less time on simple programs with simple error types but spent more time on complex programs with logical and semantic error types. In contrast, low-performing students focused more attention on easy programs with one or more syntax errors compared to high-performing students. The results of this study could shed some light on the contrasting findings of previous studies regarding fixation duration. These findings suggest that visual attention patterns of high and low-performing students may vary on multiple programs. The amount of visual effort exerted by the group depends upon the program's complexity, location of errors in the source code, type of errors injected, and the number of lines of code. This implies that the time spent finding the errors may be associated with the characteristics of the programs and the location and type of injected errors. Therefore, researchers must provide detailed information on these characteristics when describing differences in visual effort metrics between subjects engaged in bug-finding tasks.

*Key-Words:* - Visual effort metrics, fixation, visual attention, visual attention patterns, debugging, fixation metrics, program comprehension

Received: August 23, 2023. Revised: October 5, 2023. Accepted: October 16, 2023. Published: November 1, 2023.

## 1 Introduction

Program comprehension is an integral aspect of software development since other programming activities depend on it, [1]. Programmers spend a considerable amount of time comprehending source code, [2]. Unlike natural language comprehension, source code is a structured document that might be difficult to understand, [3]. Source code comprehension does not only involve understanding text structure and meaning but also understanding code execution. Programmers, therefore, need to master their ability to trace source code execution along with their ability to read its words and structures, [4].

Eye trackers have become a standard tool for conducting empirical studies in programming by recording eye-movement data of participants while performing a task to capture their visual attention, [5]. Eye trace data reveals the focus of attention and how it travels within the stimuli, providing important insight into the underlying cognitive processes of a subject, [6]. The increased use of eye-tracking data in understanding the cognitive processes of subjects while performing program comprehension tasks is based on theories about comprehension and eye movements.

The first study to employ eye-tracking data was conducted in 1990 to analyze students' comprehension processes while reading algorithms, [7]. Since then, researchers and computer science educators have been actively investigating how programmers think using eye-tracking data while they complete programming tasks such as program comprehension, [8], [9] [10], model comprehension, [11], [12], debugging, [13], [14], [15], requirements traceability, [16], [17], and collaborative programming, [18], [19]. While most of these published papers focused on program comprehension and debugging, [20], we still have limited knowledge about the visual strategies employed when finding bugs on static source codes.

Several studies have attempted to identify the differences in the visual attention patterns of experts and novices while locating bugs in static source codes using visual effort metrics, [21], [22], [23], [24]. Among the visual effort metrics, fixation counts, and fixation durations are commonly used by previous studies to determine the visual attention of participants while comprehending source code. Although no visual processing occurs during saccades, [22], used saccade lengths and fixation durations to measure the visual attention of novice and advanced programmers while identifying bugs or determining the program output.

Previous studies revealed that experts tend to focus more on areas where the error is while novices read the codes more broadly, [21], [23]. The findings imply that experts had higher fixation counts and higher fixation durations on the error lines. The study, [24], supports their findings in terms of fixation counts on the error lines, but not fixation durations on the error lines. Fixation durations and saccade lengths showed that advanced programmers had shorter fixations and saccades, [22], indicating that they can easily understand and see more details in the code. This result differs from that of, [21], [23], in terms of fixation durations, but it is consistent with, [24]. The analysis of gaze patterns of individuals in programming pairs characterized the more successful participants to have higher overall fixation counts, higher fixation counts on error lines, and longer fixation duration per program, [25]. This result is not in accordance with the findings of these studies, [22], [24].

While most of the reviewed studies agree on the results for fixation count metrics, there are differences in the findings for fixation duration metrics. Therefore, further investigations on the use of fixation metrics to determine the difference in the visual effort of experts and novices between multiple programs are required to establish a general trend. Thus, this study aimed to identify the factors affecting the varying results on fixation duration metrics and validate the results on fixation count metrics. We used visual effort metrics to determine the visual attention patterns of high and low-performing students engaged in defect-finding tasks on multiple programs. This study is similar to [21], in terms of objective, comprehension task, and analysis employed to assess the visual effort exerted by the students. Unlike the previous studies that utilized programs written in C, C++, and Python, we used programs written in Java with different numbers of lines of code, numbers of injected bugs, and cyclomatic complexity.

Exploring the visual strategies employed by experts or high-performing students when performing comprehension tasks will allow us to identify effective strategies that can be explicitly taught to low-performing students to enhance their code comprehension and debugging skills, [26].

## 2 Methodology

This paper is an analysis of a larger eye-tracking study on programmer tracing and debugging skills as well as the development of higher education's capacity to conduct eye-tracking research. The methods discussed here are also discussed in the study of, [10], [27].

### 2.1 Participants

The participants of this study were Computer Science and Management Information Systems students who have at least taken a college-level introductory programming course using Java as the programming language. A total of 64 undergraduate students from four universities in the Philippines: 16 from School A, 17 from School B, 16 from School C, and 15 from School D, participated in the eye-tracking experiment. The study used two participant groups: high-performing and low-performing. The scores of the participants in the debugging tasks were used to assign them to a particular group. High-performing group consisted of students who scored above and equal to the mean score while the low-performing group consisted of students who scored lower than the mean score.

### 2.2 Datasets

The Ateneo Laboratory for Learning Sciences previously collected the data set used in this study, which is part of a larger study on programmer eye-tracking behavior. The eye-tracking data of 64 students were collected and saved in an individual Comma-Separated Values (CSV) file. The file is composed of information regarding the fixation timestamp, the location of fixation, fixation duration, blinking count, pupil dilation, and separate values for the left and right eye movements. For the analysis of this study, only the fixation location, and fixation duration were extracted from the individual CSV file. This study used a total of 238,733 fixation data points.

## 2.3 Experimental Setup and Procedure

All participants underwent a screening process. Students were given an informed consent form to fill out and sign. Screening questionnaires were distributed to determine their eligibility to take part in the study. Students who passed the initial screening were required to undergo a nine-point eye-tracking calibration test. A written program comprehension test (20 minutes) was administered after the successful calibration test to determine the student's prior knowledge of programming. The actual eye-tracking experiment which was designed for 60 minutes followed after the written pre-test.

The Gazepoint eye tracker was used in the eye-tracking experiment with a sampling rate of 60Hz and 0.5-1 degree of accuracy. The screen resolution of the monitor was set to 1366 x 768 and the source code was presented in a full-screen window. The participants were asked to read 12 program codes with known errors and should mark the location of the errors using the mouse. There is no need for the participants to correct them. Figure 1 shows the standard setup of the eye-tracking experiment.

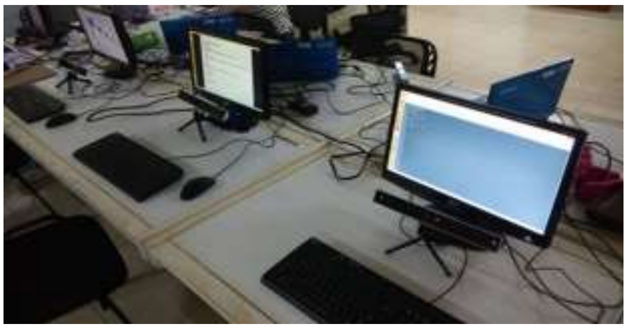


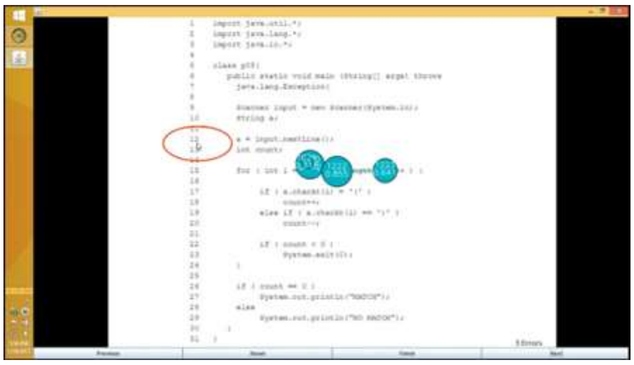
Fig. 1: Standard Set-up of the Eye Tracking Experiment

## 2.4 Hardware/Software Setup

A slide sorter program (Figure 2) with buttons Previous, Next, Reset, and Finish were created to display the specifications of the program followed by the program code with injected bugs. The Previous and Next buttons were used to navigate through the slides. The Reset button was used to clear the marked error locations of a particular slide while the Finish button was used to save the marks and end the debugging session.

## 2.5 Task Stimuli and Injected Defects

The experiment had 12 program codes as stimuli which are typically written by novice programmers. All program codes are written in Java language, contain intermediate syntax and constructs, and consist of varied lines of code. All codes presented to the participants were guaranteed to fit on the computer screen for readability and no scrolling was required.



```
import java.util.*;
import java.lang.*;
import java.io.*;

class p05{
   public static void main (String[] args) throws
     java.lang.Exception{

     Scanner input = new Scanner(System.in);
     String a;

     a = input.nextLine();
     int count;

     for ( int i = [illegible] ) {

          if ( a.charAt(i) = '(' )
               count++;
          else if ( a.charAt(i) == ')' )
               count--;

          if ( count < 0 )
               System.exit(0);
     }

     if ( count == 0 )
          System.out.println("MATCH");
     else
          System.out.println("NO MATCH");
   }
}
```

Fig. 2: Screenshot of the Slide Sorter Program

Bugs were intentionally added to the program codes. Few of these injected bugs take a minimal number of scans to detect. But quite a number take a considerable amount of time and may involve the participant's analytical skills and prior knowledge in programming. Each program was assigned either 1 or 3 bugs, with different numbers of lines of codes, cyclomatic complexity, and nested block depth as shown in Table 1.

Table 1. Code Description, Line Numbers of Injected Defects, and Metric Value of Codes

| Code | Description | Lines of Code | Line with Error | Cyclomatic Complexity | Nested Block Depth |
|---|---|---|---|---|---|
| *P01* | What's the Next Number? | 17 | 10 | 2 | 1 |
| *P02* | Reverse of Strings | 15 | 15 | 2 | 2 |
| *P03* | Arrow | 16 | 14 | 1 | 1 |
| *P04* | Q Prime | 26 | 15, 22, 29 | 6 | 4 |
| *P05* | Parenthesis Matching | 24 | 13, 17, 23 | 6 | 2 |
| *P06* | Palindrome | 19 | 15, 18, 22 | 3 | 3 |
| *P07* | Rock, Paper, and Scissor | 34 | 20, 26, 30 | 9.5 | 1.5 |
| *P08* | The Diamond Pattern | 27 | 11, 13, 18 | 7 | 2 |
| *P09* | Paralleloword | 29 | 15, 24, 30 | 7 | 3 |
| *P10* | Consecutive Words | 16 | 11, 14, 16 | 3 | 2 |
| *P11* | Earthquake's Class | 25 | 11, 18, 24 | 7 | 1 |
| *P12* | Basic Calculator | 28 | 11, 22, 27 | 5 | 1 |

Errors to locate range from the easiest to spot (syntax errors) to the hardest (semantic and logical errors). Table 2 presents the characteristics of the errors consisting of the description, type, and location of the errors injected in each program. Only 1 error was injected for programs 1-3, while programs 4-12 had 3 errors each. The injected errors were of different types and were injected in different lines and sections of the programs.

## 2.6 Experiment Procedure and Data Segmentation

The participant was asked to sit comfortably, and then the eye-tracker was adjusted to detect the eyes of the participant. Calibration of the eye-tracker on the participant was done by asking the participant to follow the white circle/red dot to check if the calibration was proper. Calibration was done since the accuracy and precision of the eye tracker data depend on successful calibration. When the calibration was successful, the participant started the experiment. The Gaze Video was closed to avoid recording the face of the participant while the experiment was ongoing. During the experiment process, the participant had to view all 12 short Java programs and locate the bugs. Out of the 12 short programs, 9 of them had 3 bugs while 3 had 1 bug each. The participant was asked to view the programs in one setting by showing the program description of each program first, followed by the program code. The eye tracker records the visual behavior of the participant while reading the static source code to find the bug/bugs in the program. A red oval appears on the screen and the participant marks the location of the errors in the program using a mouse click.

The eye tracker system continually records all the gaze movements of the participant and stores them in a CSV file format. Data such as the time of the recording (timestamp) when fixations occur, the location of the fixations (values of x and y coordinates), the fixation duration of each fixation, pupil dilation, blinking counts, and separate values for the left and right eye movements were recorded. The data from the CSV file were segmented to extract the basic values needed for the analysis.

## 2.7 Data Pre-Processing

The segmented file by stimulus viewed by each participant was used in this study. However, not all the data included in the CSV file were used in the analysis. In the context of this study, only the x and y coordinates and the fixation durations were needed for the data analysis.

The participants viewed 12 short Java programs consisting of various constructs and lines of code. Areas of Interest (AOIs) of these programs were defined based on the lines of code excluding blank lines using the OGAMA Areas of Interest module, [28], to get the AOI coordinates. In this study, a simple rectangle shape was used to mark the AOIs. The AOIs were defined per line of code to determine the visual attention exerted on each line of code, particularly the error lines.

The AOI coordinates extracted from OGAMA return coordinates with respect to the setting of the screen resolution specified when the AOIs were defined. To map the location of fixations to the program codes, the x and y coordinates from the eye-tracking data were converted by multiplying the x coordinates with 1366 and the y coordinates with 768. This was done to match the coordinates of the program codes during the experiment since the screen resolution used was 1366 x 768. In addition, the fixation durations were recorded in terms of seconds by the eye tracker and were converted into milliseconds by multiplying the duration by 1000. These processes were done for the eye-tracking data of the 64 students consisting of 238,733 data points to determine their visual attention patterns.

## 2.8 Data Analysis

To determine the difference in the visual attention patterns of high-performing and low-performing students, fixation counts and fixation durations were used. The total fixation counts on the entire stimulus (TotalFC), total fixation durations (ms) on the entire stimulus (TotalFD), total fixation counts on the error lines (FCBugLine), and total fixation durations (ms) on the error lines (FDBugLine) were computed to measure the visual effort exerted by the high and low-performing students.

Statistical analysis was conducted to compare the visual efforts of the high and low-performing students using the visual effort metrics stated above. Independent samples t-tests were used to determine whether there is statistical evidence that the visual attention patterns are significantly different between high and low-performing students. Independent samples t-tests were used on the total fixation counts, total fixation durations, total fixation counts on the error lines, and total fixation durations on the error lines of the high and low-performing students while performing debugging tasks.

## 3 Results and Discussion

The eye-tracking data of 64 students from four private universities in the Philippines were used in the analysis. Twenty-five (25) students were identified as high-performing while thirty-nine (39) students were considered low-performing based on their debugging scores. However, after data pre-processing, one (1) eye tracking data from the high-performing group was discarded since most of the fixations on the stimuli were recorded with negative x- and y- gaze coordinates and could not be mapped to the identified AOIs. Thus, only the eye gaze data of 63 students were analyzed to determine the difference in the visual attention patterns of high-performing and low-performing students.

### 3.1 Visual Attention Patterns using Total Fixation Counts

The overall fixation counts on the entire stimulus (Figure 3) show that low-performing students have higher fixation counts on programs 1-4. Programs 1-3 had 1 injected defect each and had the lowest cyclomatic complexity and number of lines of code. Further, the errors are generally simple, consisting of missing and additional semicolons, and the use of print instead of *println*. Although Program 4 had 3 injected defects, these were easy to determine since they were all syntax errors. High-performing students, on the other hand, had more fixation counts on most of the programs that are characterized by more lines of code, cyclomatic complexity, and the number of errors. Furthermore, based on the profile plot, we can see that there is a big difference between the fixation counts of high and low-performing students in programs 6, 9, and 10. These programs had 3 injected defects consisting of syntax, logical, and semantic types of errors. Program 9 had 3 logical errors located on the repetition structures, had a cyclomatic complexity of 7, and had 29 lines of code. Programs 6 and 10 had a cyclomatic complexity of 3, 19, and 16 lines of code respectively, and had combinations of syntax, semantic, and logical errors. High-performing students also had higher total fixation counts in Program 12 than the low-performing students. This program had 3 injected defects consisting of 2 logical errors and 1 semantic error.

The visual summary of the distribution and skewness of the data on the total fixation counts is presented in Figure 4 to supplement the information on the profile plot. We can see from the boxplots that several outliers are present in both groups. The boxplots show that 1 or 2 low-performing students had higher fixation counts than the maximum value of the low-performing group on several programs while 3 high-performing students in Program 2 and 1 in Program 7 had higher fixation counts than the maximum value of the high-performing group. Further, the median fixation counts of the high-performing group in Programs 6, 9, 10, and 12 show great difference from the median fixation counts of the low-performing group. The box plots also show that the data of the high-performing group are more dispersed in the majority of the programs compared to the low-performing group. Furthermore, the distribution of data for most of the programs is positively skewed for both groups.

An independent samples t-test was performed to determine if there was a significant difference in the total fixation counts of the high and low-performing students. The result of the analysis revealed that, if the type of task is not considered, there is no significant difference in the total fixation counts of the high-performing students (M = 336.50, SD = 79.60) and low-performing students (M = 293.83, SD = 90.47), t (61) = -1.901, p = 0.062. The result suggests that the total fixation counts are similar for both high and low-performing students across the different programs. This result is not surprising given that the tasks have varying levels of difficulties and presumably would require varying fixation counts as can also be inferred from the boxplots in Figure 4.

To investigate this matter more deeply, independent sample t-tests were conducted for each program to check if there were programs that would reveal statistical significance between the high and low-performing groups. The current partition of the high and low-performing groups was revised before performing the statistical tests. This was done since a student who does well overall but poorly on a specific task may incorrectly contribute “high-performing” techniques on that particular task if the current partition was retained for the analysis. Thus, the high and low performers were partitioned based on their scores in each type of task to determine the techniques that may enable a student to do well in each of the different problems. High-performing students comprise those whose scores are greater than or equal to the mean score for the given task while low-performing students comprise those whose scores are less than the mean score.

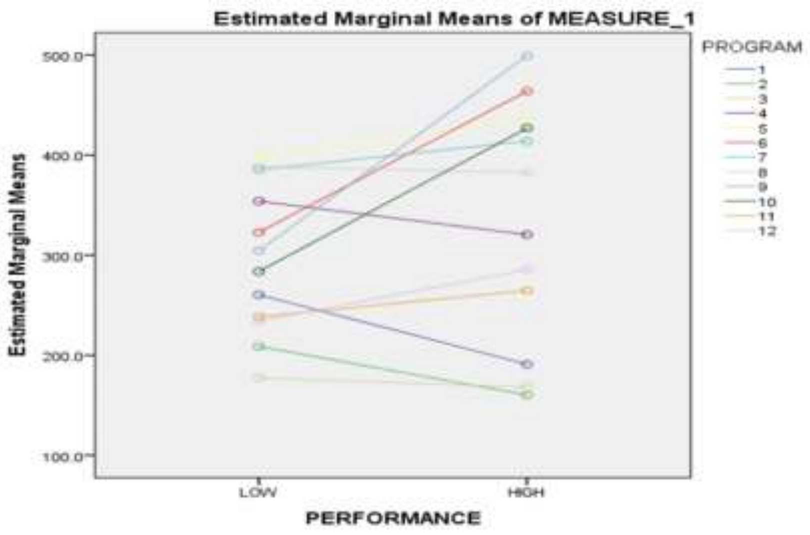


Fig. 3: Profile Plot of the Total Fixation Counts

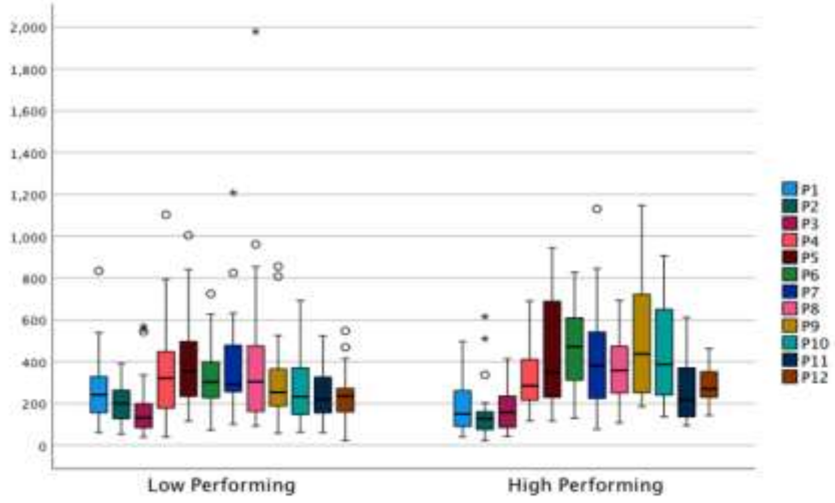


Fig. 4: Boxplots of the Total Fixation Counts

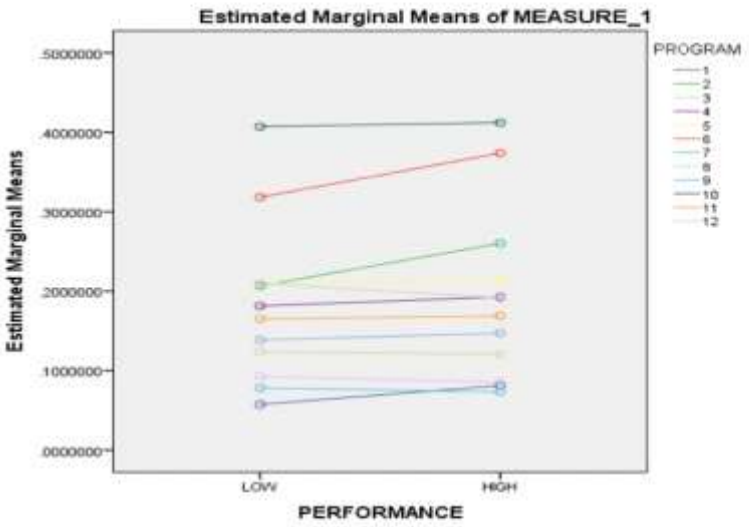


Fig. 5: Profile Plot of the Total Fixation Counts on the Error Lines

Table 3. Result of Statistical Tests on Total Fixation Counts

| Program No. | Group | Mean | Standard Deviation | t-value | p-value |
|---|---|---|---|---|---|
| 1 | Low | 271.54 | 159.04 | 1.965 | 0.054 |
| | High | 204.03 | 113.37 | | |
| 2 | Low | 215.80 | 88.68 | 1.475 | 0.145 |
| | High | 173.29 | 124.69 | | |
| 3 | Low | 195.53 | 151.75 | 1.496 | 0.141 |
| | High | 149.00 | 87.66 | | |
| 4 | Low | 374.40 | 236.78 | 1.263 | 0.211 |
| | High | 311.75 | 151.39 | | |
| 5 | Low | 393.52 | 235.78 | -0.541 | 0.591 |
| | High | 424.68 | 215.63 | | |
| 6 | Low | 320.35 | 165.40 | -1.803 | 0.076 |
| | High | 405.86 | 189.68 | | |
| 7 | Low | 355.87 | 207.60 | -1.354 | 0.181 |
| | High | 431.00 | 231.82 | | |
| 8 | Low | 375.89 | 349.03 | -0.312 | 0.756 |
| | High | 398.32 | 169.97 | | |
| 9 | Low | 375.20 | 265.72 | -0.081 | 0.936 |
| | High | 380.22 | 227.16 | | |
| 10 | Low | 307.30 | 182.08 | -1.405 | 0.165 |
| | High | 379.35 | 224.18 | | |
| 11 | Low | 254.47 | 127.67 | -0.217 | 0.829 |
| | High | 252.44 | 123.61 | | |
| 12 | Low | 237.70 | 101.69 | -1.017 | 0.313 |
| | High | 265.14 | 109.01 | | |

No significant difference was observed in the total fixation counts of the high and low-performing groups in each problem, as seen in Table 3. However, high-performing students were associated with lower fixation counts for Programs 1-4 compared to the low-performing students. These programs were injected with simple error types and have low cyclomatic complexity. Higher fixation counts were observed in high-performing students for Programs 5-12. These programs are more complex given the cyclomatic complexity, number of lines of code, number of errors, error locations, and type of injected errors. Thus, the fixation counts of the high-performing groups can be associated with the characteristics of the program codes. High-performing students exert more visual efforts on complex programs with errors that are difficult to identify such as logical and semantic errors.

Advanced programmers perform at the level of novices when the program violates the plans, and the programming discourse rules, [29]. A high number of fixations were observed from the high-performing students in Programs 5-12 since the logical and semantic errors injected in these programs violate the plan or mental schema, making it hard to comprehend the program. Repeated fixations to relevant elements in the program code are necessary to correctly identify the logical and semantic errors in the program though, a high number of eye fixations on program codes reflects an inefficient approach to finding information, [30]. The results indicate that high-performing students had more fixation counts on complex programs with logical and semantic errors, even if there was no significant difference across all programs between the high and low-performing students. Further, based on the number of fixations on all lines of code in each program, both groups had more visual attention on lines consisting of control structures, variable declarations, and compound inputs and assignment statements. This finding also suggests that these program elements can be considered as beacons that facilitate program comprehension and identification of bugs.

## 3.2 Visual Attention Patterns Using Fixation Counts on the Error Lines

The profile plot of the overall fixation counts on the error lines (Figure 5) shows that high-performing students have more fixation counts on errors injected in Programs 2 and 6 while errors injected in the majority of the programs have almost the same number of fixations from both groups. Program 2 had only one error and is located in the repetition structure. Program 6 on the other hand, had 3 errors, and two of the errors are also located in the repetition structure. These errors could be repeatedly fixed if the students follow the control flow of the program. The error in Program 2 is located on line 15 which has an additional semicolon in the for-loop structure. The buggy line of code received the highest number of fixations among all the lines in this program from both groups. However, the high-performing group had a higher percentage of fixations (27%) compared to the low-performing group (20%). The same result was observed for the number of fixations on the error lines in Program 6 where the high-performing group had a higher percentage of fixations (37%) compared to the low-performing group (32%). Figure 7 provides additional information on the distribution and skewness of the fixation counts on the buggy lines of code.

We can see from the boxplots (Figure 6) that several outliers are present in both groups. The box plots show that a greater number of extreme values from both sides of the box plots can be observed in the low-performing group than in the high-performing group. Among all the boxplots between the groups, a big difference in the median fixation counts on the buggy lines of code of the high-performing group can be observed in Program 6. The box plots also show that the data of the high-performing group are more dispersed in the majority of the programs compared to the low-performing group. Furthermore, the distribution of data is either positively or negatively skewed for both groups.

An independent samples t-test was also performed to determine if there was a significant difference in the total fixation counts on the error lines of the high and low-performing students. The result of the analysis revealed that, if the type of task is not considered, there is no significant difference in the total fixation counts on the error lines of the high-performing students (M = 0.19, SD = 0.03) and low-performing students (M = 0.18, SD = 0.03), t (42.24) = -1.644, p = 0.108. The result suggests that the total fixation counts on the error lines are similar for both high and low-performing students across different programs. Additional statistical tests were performed since there was no statistically significant difference between the groups across the different programs. Independent samples t-tests were conducted for each program to distinguish between a high-performing student working on a difficult task versus a low-performing student working on an easy task or vice versa. The partition of high and low-performing students used in the statistical tests of fixation counts per problem was also used in this analysis.

Only Programs 1 and 6 revealed a significant difference between the groups, as shown in Table 4. The high-performing students were associated with statistically significantly higher total fixation counts on the error lines of these programs than the low-performing students. The number of fixations on an AOI can be linked to its importance, [31]. Although there is no significant difference in the visual attention of high and low-performing students in terms of the fixation counts on the error lines across all programs, the result suggests that high-performing students focused more on the control structure elements of the program code. Thus, more visual attention can be observed from the high-performing students on errors located in the repetition structures, particularly in Program 6. Further, we can also observe that high-performing students had more fixation counts on errors in 9 out of 12 programs. Low-performing students had slightly higher fixation counts on errors in Programs 3, 10, and 12. This result is in line with the findings of, [21], [23], [24], that experts or advanced programmers have more fixation counts on the buggy lines of code.

Table 4. Result of Statistical Tests on Total Fixation Counts on the Error Lines

| Program No. | Group | Mean | Standard Deviation | t-value | p-value |
|---|---|---|---|---|---|
| 1 | Low | 0.04 | 0.03 | -3.425 | 0.001* |
| | High | 0.09 | 0.06 | | |
| 2 | Low | 0.22 | 0.10 | -0.472 | 0.638 |
| | High | 0.23 | 0.16 | | |
| 3 | Low | 0.13 | 0.06 | 0.402 | 0.689 |
| | High | 0.12 | 0.06 | | |
| 4 | Low | 0.17 | 0.07 | -1.147 | 0.256 |
| | High | 0.20 | 0.08 | | |
| 5 | Low | 0.21 | 0.05 | -2.252 | 0.802 |
| | High | 0.21 | 0.06 | | |
| 6 | Low | 0.29 | 0.09 | -3.076 | 0.003* |
| | High | 0.37 | 0.09 | | |
| 7 | Low | 0.07 | 0.04 | -0.433 | 0.667 |
| | High | 0.08 | 0.03 | | |
| 8 | Low | 0.20 | 0.06 | -0.69 | 0.493 |
| | High | 0.21 | 0.08 | | |
| 9 | Low | 0.13 | 0.05 | -1.611 | 0.112 |
| | High | 0.15 | 0.04 | | |
| 10 | Low | 0.41 | 0.14 | 0.362 | 0.719 |
| | High | 0.40 | 0.12 | | |
| 11 | Low | 0.16 | 0.04 | -0.315 | 0.754 |
| | High | 0.17 | 0.04 | | |
| 12 | Low | 0.09 | 0.03 | 0.51 | 0.612 |
| | High | 0.09 | 0.03 | | |

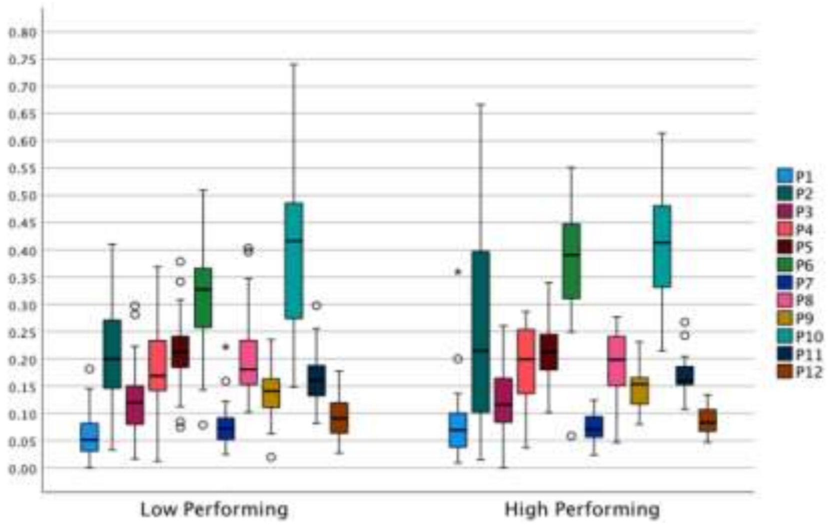


Fig. 6: Boxplots of the Total Fixation Counts on the Error Lines

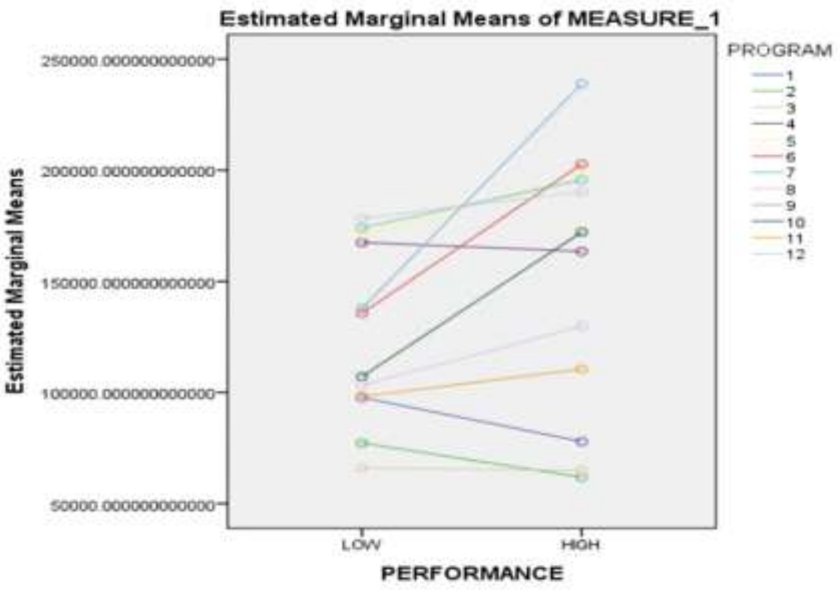


Fig. 7: Profile Plot of the Total Fixation Durations

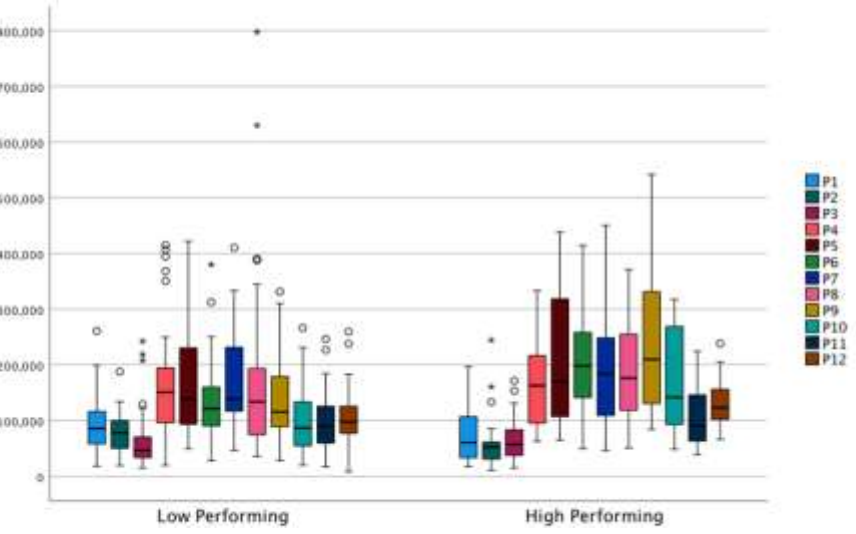


Fig. 8: Boxplots of the Total Fixation Durations

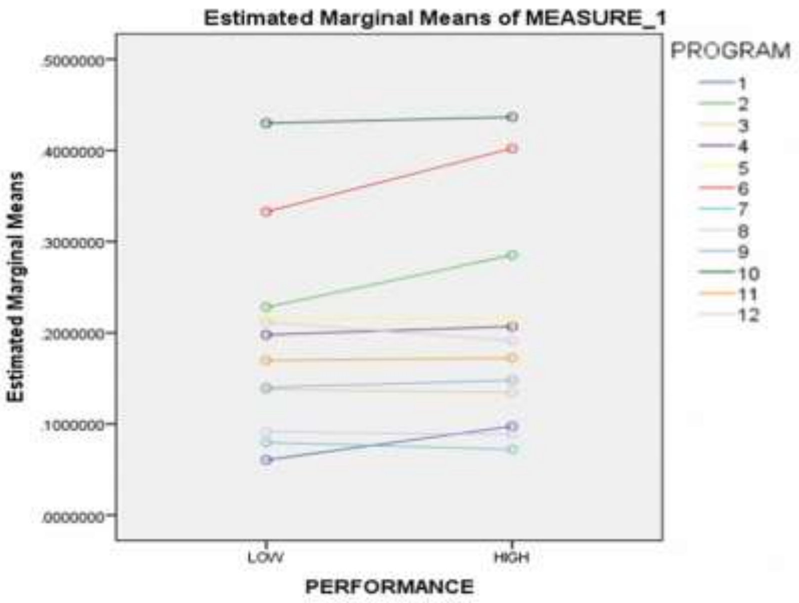


Fig. 9: Profile Plot of the Total Fixation Durations on the Error Lines

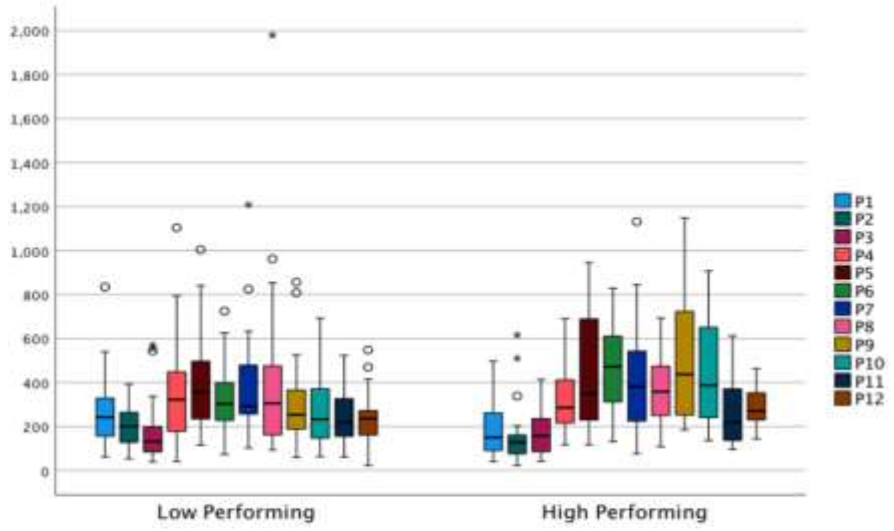


Fig. 10: Boxplots of the Total Fixation Durations on the Error Lines

## 3.3 Visual Attention Patterns using Total Fixation Durations

Figure 7 shows the profile plot of the total fixation durations of the high and low-performing students while Figure 8 shows the visual summary of the distribution and skewness of the data on total fixation durations. Based on the profile plot, high-performing students spent more time on programs with more lines of code, code complexity, number of errors, and programs with logical and semantic error types. Low-performing students, on the other hand, spent more time reading Programs 1-4. We note that Programs 1-3 had only one error each and Program 4 had 3 syntax errors, which can be considered as simple programs with simple error types.

The boxplots in Figure 8 show that several outliers are present in both groups. The boxplots show that there are extreme values on all programs in the low-performing group while 3 out of 12 programs had extreme values in the high-performing group. Further, the median fixation durations of the high-performing group in Programs 6, 9, 10, and 12 show great difference from the median fixation durations of the low-performing group. The box plots also show that the data of the high-performing group are more dispersed in the majority of the programs compared to the low-performing group. Furthermore, the distribution of data for many of the programs is positively skewed for both groups.

An independent samples t-test was performed to determine if there was a significant difference in the total fixation durations between the high and low-performing students. The result of the analysis revealed that there is a significant difference in the total fixation durations of the high-performing students (M = 151063.59, SD = 36731.93) and low-performing students (M = 125107.42, SD = 41689.59), t (61) = -2.508, p = 0.015. The result suggests that the total fixation durations of the high-performing students are significantly higher than the low-performing students across the different programs. This result has been published in, [10], as a preliminary analysis to check for the difference in the fixation duration of high and low-performing students and to analyze if the difference has a relationship with the pattern similarity. However, the result presented in this paper discussed how the complexity and difficulty of the programs and the debugging task affect the visual attention patterns of the students.

Fixation durations are influenced by the complexity and difficulty of the visual content and task being performed, [22], [31]. More visual effort may be exerted for complex programs and programs with errors that are difficult to identify such as logical and semantic errors. This finding is consistent with, [22], since high-performing students have shorter fixation durations on Programs 1-4. This could mean that high-performing students can understand the code easily and be able to see more details in it compared to low-performing students. However, high-performing students spent more time on Programs 5-12 which consisted of a greater number of errors, lines of code, and complex programming constructs, and had combinations of logical, semantic, and syntax errors. The complexity of the program and the difficulty in identifying the errors injected in the programs may have caused low-performing students to stop trying to answer the problems and move to other programs, resulting in lower fixation durations. High-performing students, on the other hand, employ more careful viewing of the program codes to effectively identify the errors resulting in higher fixation durations. As shown in Figure 7, there is a big difference in the fixation durations of the groups on Programs 6, 9, and 10. These programs had complex programming constructs and were injected with logical and semantic errors that were difficult to identify. Therefore, the difference in the overall fixation durations between the groups is influenced by the program constructs and the errors injected into the programs. High-performing students spent less time on simple programs with simple error types but spent more time on complex programs with logical and semantic error types.

## 3.4 Visual Attention Patterns using Fixation Durations on the Error Lines

An independent samples t-test was also performed to determine if there was a significant difference in the total fixation durations on the error lines between the high and low-performing students. The result of the analysis revealed that there is no significant difference in the total fixation durations on the error lines of the high-performing students (M = 0.20, SD = 0.04) and low-performing students (M = 0.19, SD = 0.03), t (42.24) = -1.614, p = 0.112. The result suggests that the total fixation durations on the error lines are similar for both high and low-performing students across the different programs. Additional statistical tests were performed since no statistically significant difference was observed across the different programs. Independent samples t-tests were conducted for each program to check if there were programs that would reveal statistical significance between the high and low-performing groups. The partition of high and low-performing students used in the previous sections was also used in the statistics tests. Similar to the findings on fixation counts on the error lines, only Programs 1 and 6 revealed significant differences between the groups (Table 5). The high-performing students were associated with statistically significantly higher total fixation durations on the buggy lines of code of Programs 1 and 6 than the low-performing students.

The increased fixation duration in AOIs of the visual stimulus could be used to detect more difficult-to-process components or AOIs that are engaging the cognitive resources of the observer, [31]. Experts tend to focus more on areas where the errors are located while novices read the codes more broadly, [21], [23]. In contrast with the findings of the latter, novices spent more time on the buggy lines of code, [22], [24]. However, based on the profile plot in Figure 9, fixation durations on the buggy lines of code are almost similar for both high and low-performing students except for Programs 2 and 6. Refer also to the boxplots in Figure 10 to see the visual summary of the distribution and skewness of the total fixation durations on the error lines of the high and low-performing students. The boxplots are similar to the boxplots presented in Figure 6 show the visual summary of the distribution and skewness of the total fixation counts on the error lines of the high and low-performing students.

Table 5. Result of Statistical Tests on Total Fixation Durations on the Error Lines

| Program No. | Group | Mean | Standard Deviation | t-value | p-value |
|---|---|---|---|---|---|
| 1 | Low | 0.04 | 0.03 | -3.578 | 0.001* |
| | High | 0.10 | 0.09 | | |
| 2 | Low | 0.24 | 0.11 | -0.532 | 0.597 |
| | High | 0.26 | 0.18 | | |
| 3 | Low | 0.14 | 0.08 | -0.006 | 0.995 |
| | High | 0.14 | 0.08 | | |
| 4 | Low | 0.18 | 0.10 | -1.229 | 0.224 |
| | High | 0.22 | 0.11 | | |
| 5 | Low | 0.21 | 0.06 | -0.401 | 0.69 |
| | High | 0.22 | 0.07 | | |
| 6 | Low | 0.30 | 0.11 | -3.096 | 0.003* |
| | High | 0.39 | 0.11 | | |
| 7 | Low | 0.08 | 0.05 | 0.094 | 0.926 |
| | High | 0.08 | 0.03 | | |
| 8 | Low | 0.20 | 0.08 | -0.712 | 0.479 |
| | High | 0.21 | 0.08 | | |
| 9 | Low | 0.14 | 0.05 | -1.316 | 0.193 |
| | High | 0.15 | 0.04 | | |
| 10 | Low | 0.43 | 0.16 | 0.199 | 0.843 |
| | High | 0.43 | 0.13 | | |
| 11 | Low | 0.17 | 0.05 | 0.26 | 0.795 |
| | High | 0.17 | 0.05 | | |
| 12 | Low | 0.09 | 0.04 | -0.157 | 0.876 |
| | High | 0.09 | 0.03 | | |

High-performing students spent more time looking at the buggy lines of 9 out of 12 programs. Errors in the programs located in the repetition structure (for loop) which is a complex programming construct since it consists of initialization, condition, and increment/decrement operation may be difficult to process. Thus, increased fixation duration was observed. The result of this study supports previous study findings, [21], [23], that high-performing students or experts spent more time on the error lines but only for Programs 1 and 6. Overall, the result shows that there is no significant difference in the fixation durations on the error lines between the high and low-performing students. The contrasting results regarding the findings in the fixation durations on the buggy lines of code may be associated with the characteristics of the programs used in their studies and the type and location of errors injected in the programs. Therefore, when describing the difference in fixation durations on the buggy lines of code between groups, it might be necessary to provide information on the characteristics of the program and the injected errors.

The difference in the visual attention of high and low-performing students while performing a debugging task can be measured using fixation count and fixation duration metrics. The findings in the analysis of visual attention suggest that longer fixation durations and more fixation counts indicate difficulty in identifying the buggy lines of code. High-performing students exert more visual attention on complex programs with logical and semantic errors resulting in more fixation counts and longer fixation durations compared to low-performing students. This visual behavior of high-performing students may be related to the field-independent (FI) cognitive style theory of human cognition. The FI individuals tend to choose a more analytical processing approach and they pay attention to details, [32]. Conversely, the visual behavior of low-performing students may be related to field-dependent (FD) cognitive style wherein they choose a more holistic way of processing visual information and experience difficulties in identifying details in the complex visual stimulus. FI individuals have significantly more and longer fixations than FD individuals and these differences may be associated with the analytical nature of FI individuals.

## 4 Conclusion

This study contributes to the evidence of the effectiveness of eye tracking as a method to enrich computing education research. The analysis of the visual effort metrics provided considerable insights into the visual attention patterns of high and low-performing students in finding source code defects. Results revealed that high and low-performing students could be distinguished based on their visual attention patterns. High-performing students spend more time on complex programs with logical and semantic errors to effectively find source code defects. On the other hand, low-performing students spend more time on simple programs with simple error types. The result of the difference in the time spent in finding source code defects suggests that high-performing students prefer a more analytical approach while low-performing students choose a more holistic approach. Our findings also revealed that both high and low-performing students had more visual attention on lines consisting of control structures, variable declarations, and compound inputs and assignment statements. These program elements can be considered as beacons that facilitate program comprehension and identification of source code defects. These findings suggest that visual attention patterns of high and low-performing students may vary on multiple programs. The amount of visual effort exerted by the group depends upon the program's complexity, location of errors in the source code, type of errors injected, and the number of lines of code. This implies that the time spent finding the errors may be associated with the characteristics of the programs and the location and type of injected errors. Therefore, researchers must provide detailed information on these characteristics when describing differences in visual effort metrics between subjects engaged in bug-finding tasks.

By exploring the visual strategies employed by the high-performing students using eye-tracking data, we could develop learning materials and activities that could help low-performing students improve their code reading and debugging skills. Debugging should be taught as a program comprehension or program understanding task rather than a search task. Students should learn how to identify the relevant code elements of the program to identify source code defects. Programming educators should teach more problem-solving activities to develop the student's analytical skills. Further, teaching students to consciously employ debugging strategies according to code size, program structure, and type of errors would enhance their ability to plan for the debugging task.

*Acknowledgement:*
The author would like to thank the Ateneo de Manila, Ateneo de Davao, University of Southeastern Philippines, University of San Carlos, 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”. The author also thanks Prof. Maria Mercedes Rodrigo for being her mentor in conducting this study.

*References:*
[1] Sharma, K., Jermann, P., Nüssli, M., and Dillenbourg, P. (2012). Gaze Evidence for Different Activities in Program Understanding. In *Proceedings of 24th Workshop of the Psychology of Programming Interest Group, PPIG*, pp.20-31.

[2] Schröter, I., Krüger, I., Siegmund, 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, pp.308-311.

[3] Tvarozek, J., Konopka, M., Navrat, P., and Bielikova, M. (2016). Studying Various Source Code Comprehension Strategies in Programming Education. *In Proceedings of the Third International Workshop on Eye Movements in Programming: Models to Data*, pp.25-26.

[4] 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*, pp.255-265.

[5] Sharafi, Z., Soh, Z., Guéhéneuc, Y., and Antoniol, G. (2012). Women and men—Different but equal: On the impact of identifier style on source code reading. *In Proceedings of the 2012 IEEE 20th International Conference on Program Comprehension (ICPC), IEEE Explore*, pp.27–36.

[6] Begel, A. and Vzrakova, H. (2018). Eye movements in code review. In Proceedings of the Workshop on Eye Movements in Programming (EMIP '18). (Poland, 2018), ACM, pp.1-5. https://doi.org/10.1145/3216723.3216727

[7] Crosby, M. E. and Stelovsky, J. (1990). How do we read algorithms? A case study. *Computer* 23, 1 (1990), pp.25-35.

[8] Jbara, A. and Feitelson, D. G. (2017). How programmers read regular code: A controlled experiment using eye-tracking. *Empirical Software Engineering*, 22(3), pp.14440-1477.

[9] Peachock, P., Iovino, N., And Sharif, B. (2017). Investigating Eye Movements in Natural Language and C++ Source Code—A Replication Experiment. *In Proceedings of the 11th International Conference on*

*Augmented Cognition. Neurocognition and Machine Learning (AC'17), Springer International Publishing*, pp.206–218.

[10] Tablatin, C. L. S., & Rodrigo, M. M. T. (2022). Identifying Code Reading Strategies in Debugging using STA with a Tolerance Algorithm. *APSIPA Transactions on Signal and Information Processing*, Vol.11(1).

[11] Cagitlay, N. E., Tokdemir, G., Kilic, O., and Topalli, D. (2013). Performing and analyzing non-formal inspections of entity relationship diagram (ERD). *Journal of Systems and Software*, 86(8), pp.2184–2195.

[12] Jeanmart, S., Guéhéneuc, Y-G., Sahraoui, H., and Habra, N. (2009). Impact of the visitor pattern on program comprehension and maintenance. *In Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement, IEEE Computer Society*, pp.69–78.

[13] Barik, T., Smith, J., Lubick, K., Holmes, E., Feng, J., Murphy-Hill, E., and Parnin, C. (2017). Do developers read compiler error messages?. *In Proceedings of the 39th International Conference on Software Engineering, IEEE Press*, pp.575–585.

[14] Lin, Y., Wu, C., Hou, T., Lin, Y., Yang, F., and Chang, C. (2016). Tracking students cognitive processes during program debugging: an eye-movement approach. *IEEE Transactions on Education*, 59(3), pp.175–186.

[15] Chen, M. and Lim, V. (2013). Eye gaze and mouse cursor relationship in a debugging task. *In HCI International 2013—Posters Extended Abstracts, Springer*, pp.468-472.

[16] Ali, N., Sharafl, Z., Gueheneuc, Y-G., and Antoniol, G. (2012). An empirical study on requirements traceability using eye-tracking. *28th IEEE International Conference on Software Maintenance (ICSM), (Italy, 2012), IEEE*, pp.191-200.

[17] Walters, B., Shaffer, T., Sharif, B., And Kagdi, H. (2014). Capturing software traceability links from developer's eye gazes. *In Proceedings of the 22nd International Conference on Program Comprehension (ICPC'14), ACM*, pp.201–204.

[18] Villamor, M. And Rodrigo, M. M. (2017). Characterizing 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.

[19] Villamor, M. M., & Rodrigo, M. M. T. (2022). Predicting Pair Success in a Pair Programming Eye Tracking Experiment Using Cross-Recurrence Quantification Analysis. *APSIPA Transactions on Signal and Information Processing*, Vol.11(1).

[20] Obaidellah, U., Al Haek, M., and Cheng, P. C-H. (2018). A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Computing Surveys*, 51 (1), 5:1-5:58.

[21] 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), pp.169-175.

[22] 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)*, pp.613-620.

[23] Sharif, B., Falcone, M. and Maletic, J. I. (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 (ETRA'12), ACM*, pp.381-384.

[24] 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). (Florida, 2014) ACM*, NY, USA, pp.231-234.

[25] Villamor, M. And Rodrigo, M. M. (2019). Analyzing Gaze Patterns of Individuals in Programming Pairs. *In Proceedings of the Philippine Computing Science Congress 2019.*

[26] Bednarik, R., Busjahn, T., and Schulte, C., (Eds.). (2014). Eye Movements in Programming Education: Analyzing the Expert's Gaze. *In Proceedings of the First International Workshop*, Finland, 2014, pp.1-3.

[27] Tablatin, C. L., & Rodrigo, M. M. (2018). Identifying Common Code Reading Patterns using Scanpath Trend Analysis with a Tolerance. *In Proceedings of thee 26th International Conference for Computers in Education (ICCE 2018)*, Metro Manila, Philippines.

[28] Von Mayrhauser, A., and Lang, S. (1999). A coding scheme to support systematic analysis of software comprehension. *IEEE Trans. on Software Engineering*, 25(4), pp.526–540.

[29] Soloway, E. and Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transaction on Software Engineering*, Vol. SE-10, No. 5, 595-609.

[30] Goldberg, J. H. and Kotval, X. P. (2010). Computer interface evaluation using eye movements: methods and constructs, *International Journal of Industrial Electronics*, 24(6), pp.631-645.

[31] Bylinskii, Z., Borkin, M. A., Kim, N. W., Pfisher, H., and Oliva, A. (2015). Eye fixation metrics for large-scale evaluation and comparison of information visualizations. *In Eye Tracking and Visualization, eds M. Burch, L Chuang, B. Fisher, A. Schmidt, and D. Weiskopf (Cham: Springer)*, 235-255. Doi: 10.1007/978-3-319-47024-5_14

[32] Raptis, G. E., Katsini, K., Belk, M., Fidas, C., Samaras, G., and Avouris, N. (2017). Using Eye Gaze Data and Visual Activities to Infer Human Cognitive Styles: Method and Feasibility Studies. *In Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization (UMAP '17). (Bratislava Slovakia, 2017), ACM*, NY, USA, pp.164-173.

## Contribution of Individual Authors to the Creation of a Scientific Article

The sole author of this scientific article independently conducted and prepared the entire work from the formulation of the problem to the final findings and solution.

**Sources of Funding for Research Presented in a Scientific Article or Scientific Article Itself**

This study was supported in part by the 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”.

## Conflict of Interest

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