Kashihara, A. et al. (Eds.) (2024). Proceedings of the 32nd International Conference on Computers in Education. Asia-Pacific Society for Computers in Education

Novice Programmers' Saccadic Patterns in Error Message Comprehension and Syntax Error Identification

Caren PACOL[ab*], Maria Mercedes RODRIGO[a] & Christine Lourrine TABLATIN[b]
[a]*Ateneo de Manila University, Philippines*
[b]*Pangasinan State University, Philippines*
*ambat_caren@yahoo.com

Abstract: Learning debugging is crucial for novice programmers, and as compiler error messages play a vital role in this process, exploring how novice programmers read and interpret these messages has become a focal point for researchers in computer science education. Understanding how novice programmers interpret compiler error messages is essential, as debugging skills are fundamental for their development. In line with this, our prior works investigated how student programmers process syntax errors embedded in Java and C++ programs and the relationship between novice programmers' personalities and their visual attention patterns using fixation-based metrics. Our previous studies were limited in that they did not capture the dynamics of the participants' eye movements. As our next step, in this study, we present saccade-based metrics that we used to gain insights into the problem-solving strategies employed by novice programmers. We found that high and low performing participants demonstrate similar strategies for scanning through code and detecting errors. Furthermore, high performing participants demonstrate longer average saccade lengths, indicative of purposeful exploration and the gathering of more information in each eye movement. Additionally, our findings indicate that high performing participants exhibit a lower average regression ratio, suggesting a greater ability to maintain focus and comprehension throughout problem-solving tasks. In future work, integrating contextual factors such as task complexity, familiarity with programming languages and individual learning styles into the analysis of saccade-based metrics can help identify situational factors shaping problem-solving strategies.

Keywords: Debugging, Eye tracking, Novice programmers, Saccades

1. Introduction

Compiler error messages assist beginner programmers in identifying and correcting syntax errors while debugging. These messages can take the form of either literal or non-literal. Literal syntax error messages are generated by the compiler to precisely match the specific error encountered. For instance, if the error message states 'semicolon (;) expected,' a semicolon added to the specified line would resolve the issue if the error were literal. Conversely, a non-literal error message does not accurately depict the actual mistake (Dy & Rodrigo, 2010). In the same scenario, if 'semicolon (;) expected,' were a non-literal error, adding a semicolon to the indicated line would not fix the problem since the root cause of the error lies elsewhere.

Given the significance of debugging within computer science and its related fields, along with the crucial role compiler error messages hold in this process, there is an increasing focus among researchers in computer science education on investigating how novice programmers interpret and respond to these messages, and how these approaches differ based on the proficiency level of the programmer.

This study is a continuation of a larger research project on the use of eye tracking to determine differences between students of different ability levels and characteristics. The first study by Tablatin and Rodrigo (Tablatin & Rodrigo, 2023) made use of eye tracking to determine the visual attention of high- and low-performing students and found that they

differed when reading and acting upon compiler error messages. The study found that high-performing students exerted more visual attention on the buggy lines identified by the compiler error messages. A subsequent study by Pacol, Rodrigo, and Tablatin (2024) on the same dataset investigated how student visual attention varied, depending on whether the error was literal or non-literal. The study found that the type of error introduced in the program can affect how high and low performing students visually attend to compiler error messages and error lines. A later study by Pacol, Rodrigo, and Tablatin (2024) investigated the relationship between novice programmers' personalities and their visual attention patterns. They found that students with high levels of agreeableness and conscientiousness tend to allocate minimal visual attention to compiler error messages and error lines, respectively.

The main limitation of these prior works is that the analysis was limited to fixation counts and fixation durations, a subset of what Sharafi et al. (2020) calls third-order data. As our next step, we processed additional third-order data metrics, specifically related to saccades. Studying saccade-based metrics provides valuable insights into how individuals process visual information during problem-solving. These metrics, such as saccade counts, saccade rates, saccade ratio, saccade length and regressions, can reveal patterns in how the eyes move between relevant areas, such as code errors and corresponding error messages in programming tasks. Our research questions include:

1. How do saccadic patterns to compiler error messages differ between high and low-performing novice programmers?
2. How do saccadic patterns to error lines differ between high- and low-performing novice programmers after their first fixation on compiler error messages?
3. How does average saccade length differ between high and low performing novice programmers?
4. How does average regression rate differ between high and low performing novice programmers?

1.1 Saccades and Associated Metrics

A saccade refers to an eye movement that shifts from one fixation point to another, lasting between 30 and 80 milliseconds (Holmqvist et al., 2011), commonly linked with visual information exploration or navigation (McChesney & Bond, 2020). A study by Busjahn et al. (2015) introduced two categories of saccades: regressive saccades, which denote backward eye movements, and forward saccades (Aljehane, 2022; Busjahn et al., 2015). Forward saccades are saccades in the direction of the target's movement (Goettker et al., 2019). For instance, when reading a story written in the English language, forward saccades can involve left-to-right eye movements in horizontal saccades or downward/upward eye movements in vertical saccades. We defined forward saccades as saccades to lines of code that were not previously visited and regressive saccades as saccades to lines of code that were already previously visited. Through the analysis of eye movements utilizing metrics related to regressive saccades, we can distinguish novice programmers who exhibit a higher frequency of regressive saccades. Moreover, an increased frequency of backward saccades could serve as a crucial indicator of challenges and confusion in comprehending a task (Poole & Ball, 2005).

In this study, we presented saccade-based metrics by taking into account five elements: the saccade count, saccade ratio, saccade rate, saccade length and regression ratio. We defined these elements as follows.

Saccade count refers to the total number of rapid eye movements made within a specified stimulus or area of interest within a stimulus.

Saccade Ratio indicates the proportion or ratio of saccades before a first fixation within the area of interest to the total number of saccades made during stimulus processing. Dividing the number of saccades made before the first fixation within the area of interest by the total number of saccades made during the entire stimulus processing indicates how quickly the student noticed the area of interest. We compute the ratio to normalize the data across different students and stimuli. This process allowed for a fair comparison across different

stimulus conditions and accounted for variations in overall saccadic activity. The saccade ratio directly relates to when the participant first looked at the AOI in relation to their saccadic movements.

Saccade rate (or saccade frequency) is the number of saccadic eye movements per unit time (Ohtani, 1971; Mahanama et al., 2022). We calculated saccade rate by dividing the number of saccades directed towards the AOI by the total duration of time participants spent viewing the entire stimuli. This calculation provides a measure of how often participants' gaze shifts towards the AOI of interest within a given time frame.

Saccade length refers to the distance between successive fixations of the participant (Busjahn et al., 2015).

Regression ratio can be determined by counting the regressive saccades and dividing the number of regressive saccades by the total number of saccades in the stimuli (Poole and Ball 2005; Busjahn et al. 2011; Sharafi et al., 2020). We opted to use the terminology 'regression ratio' consistently throughout this paper to maintain internal coherence in our analysis. It's worth noting that in some existing literature, a similar metric may be referred to as the 'regression rate.' By adopting the term 'regression ratio' in our study, we aim to ensure clarity and consistency in our discussion and interpretation of saccadic behavior.

2. Methodology

The study included college-level students, with a total of 63 participants: 31 from School A in Metro Manila and 32 from School B in Pangasinan. The data from 12 participants from School A and 6 participants from School B were excluded due to insufficient fixation recordings or frequent head movements, resulting in inaccurate fixation data collection. Consequently, the analysis was conducted using data from 19 students from School A and 26 students from School B, resulting in a dataset of 45 students. While age and gender information were not recorded, as some participants chose not to disclose these details in the survey administered during the eye-tracking experiment, all participants had introductory programming proficiency, having completed at least one programming course.

Two separate sets of stimuli were employed, each containing the same problems. The stimuli, code complexity, and experimental procedure followed the guidelines outlined in Tablatin and Rodrigo (2023). Data sources and pre-processing steps aligned with the methodologies detailed in Tablatin and Rodrigo (2023) and Pacol et al. (2024).

2.1 Data Analysis

We classified students into high and low performers based on their total scores across Problems 1 to 5. Participants with scores at or above the average were categorized as high performers, and those below as low performers. School A had 12 high performers and 7 low performers, while School B had 15 and 11, respectively.

We devised a python program to automatically calculate saccade metrics, including count, ratio, rate, length, and regression ratio, for both compiler error messages and error lines, as well as for the entire stimuli in each of the five programs. The program consolidates participants' saccade data into CSV files. With five problems presented, data from participants at School A and School B were saved in five separate CSV files. These were then summarized into a single CSV file.

We have included in our counting of saccades the following categories: Vertical Next, which refers to forward saccades that move one line down (Busjahn et al., 2015); Vertical Later, pertaining to forward saccades that either stay on the same line or move down any number of lines (Busjahn et al., 2015); and Horizontal Later, defined as forward saccades within a line (Busjahn et al., 2015). Additionally, we included in our analysis Horizontal Earlier, described as backward saccades within a line (McChesney & Bond, 2020); Vertical Previous, referring to backward saccades that move one line up (McChesney & Bond, 2020); and Vertical Earlier, which McChesney & Bond (2020) defined as backward saccades that move up any number of lines. We presented the saccade metrics that we used to answer each of

the four research questions in the following subsections. The calculations of the saccade metrics are described and illustrated using Figure 1. Given Figure 1, the yellow dots are the fixations while the red lines are the saccades. The labels of the dots represent the sequence of visits and fixation durations while the labels of the lines represent the saccade lengths. The sequence of visits is A B C D E F.

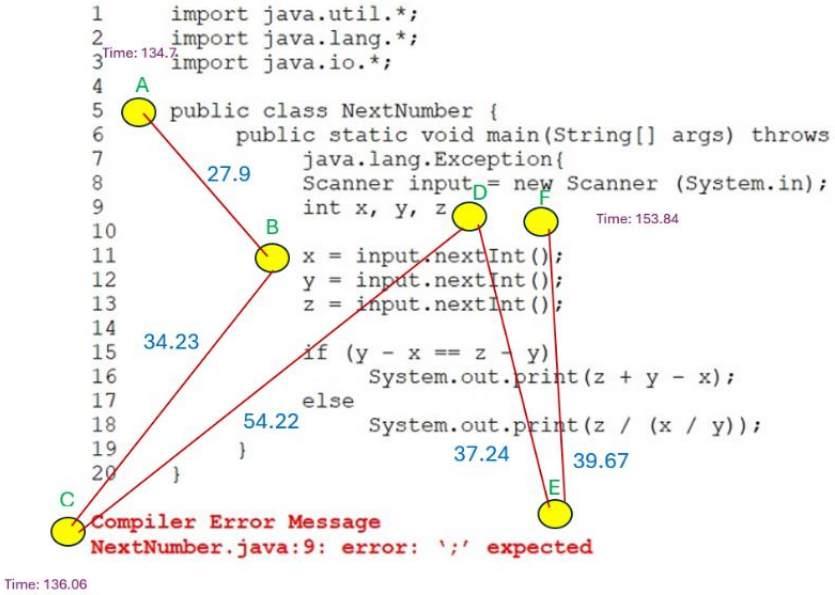

Figure 1. Visualization of saccade metrics.

2.1.1 Saccadic Patterns to Compiler Error Messages of High and Low Performers

Programming teachers train students to comprehend and deal with compiler error messages to quickly identify and address issues in their code, leading to more efficient programming. At the very least, one would think that more proficient student programmers would pay attention to the compiler error message before finding the line containing the syntax error (error line) in the program. To determine if this hypothesis is true, we counted the saccades before a first fixation on the compiler error message in each of the five programs and determined when the participant first looked at the compiler error message using Equation 1. Saccades referred to in this case are saccades on other lines in the program other than the compiler error message. A higher saccade ratio on compiler error messages may suggest that participants are actively searching or exploring the stimuli. This behavior may indicate an initial stage of visual scanning and exploration to locate relevant information. We calculated the saccade rate on compiler error messages using Equation 2 to measure how frequently participants' gazes shifted towards them. A higher saccade rate suggests that the compiler error message is perceived as salient during bug-finding. We also averaged the saccade ratio, saccade rate, and regression ratio on compiler error messages in all five programs.

Eq. 1. Saccade Ratio (Compiler Error Message) = saccade counts before a first fixation on compiler error message / total saccade counts in the entire slide

Eq. 2. Saccade Rate (Compiler Error Message) = number of saccades to compiler error message / total durations of viewing the entire slide

As illustrated in Figure 1, there are 2 saccades before a fixation to the compiler error message and these are A-B and B-C. A-B, B-C, and C-D are forward saccades while D-E and E-F are regressive saccades. The saccade ratio for compiler error messages is calculated by dividing the two saccades before fixation to the compiler error message by the total number of saccades, equivalent to five. This calculation yields a ratio of 0.40. The saccade rate for a compiler error message is calculated by dividing the number of saccades to the compiler error message, which is equivalent to 2, by the total viewing time of the entire program. This viewing time is computed by subtracting the beginning timestamp (134.7) from the end timestamp

(153.84) of the stimuli. Thus, the saccade rate for compiler error messages is 0.10 saccades per second.

2.1.2 Saccadic Patterns to Error Lines of High and Low Performers

We examined how novice programmers respond to compiler error messages by counting the saccades before their first fixation on the error line after their first fixation on a compiler error message. This saccade count helped us calculate the saccade ratio (Equation 3) and saccade rate (Equation 4) for error lines. We then averaged these metrics across all five programs. High performers are expected to show a higher saccade rate and lower saccade ratio, suggesting efficient visual processing, with fewer unnecessary fixations around the error lines.

Eq. 3. Saccade Ratio (error lines $_{\text{after first fixation on compiler error message}}$) = saccade counts before a first fixation on error line after first fixation on compiler error message / total saccade counts in the entire slide after a first fixation on the compiler error message

Eq. 4. Saccade Rate (error lines $_{\text{after first fixation on compiler error message}}$) = number of saccades on error line / total duration of viewing the entire slide after a first fixation on the compiler error message

In Figure 1, when considering L9 as the error line, the number of saccades before fixation to the error line after the first fixation on the compiler error message is 1 (C-D). After the first fixation on the compiler error message, the total saccade count to the error line is 3. Consequently, the saccade ratio for the error line after the first fixation on the compiler error message is 0.33.

2.1.3 Saccade Lengths of High and Low Performers

We calculated the saccade length between two consecutive fixations given their x and y coordinates using Euclidean distance formula. We first computed the difference in x and y coordinates using Equations 5 and 6. Then we calculated the Euclidean distance using Equation 7 to yield the saccade length. Next, we used Equation 8 to calculate average saccade length in the entire program. Finally, we computed the average of average saccade lengths in all five programs.

Eq. 5. $\Delta x = x2 - x1$

Eq. 6. $\Delta y = y2 - y1$

Eq. 7. Saccade Length = $\sqrt{(\triangle x\text{^}2 + \triangle y\text{^}2)}$

Eq. 8. Average Saccade Length = $\sum$saccade lengths / number of consecutive fixations

In Figure 1, the average saccade length of all saccades made in the stimuli is 38.65 pixels.

2.1.4 Regression Ratio of High and Low Performers

We counted regressive saccades based on the number of revisits in each line. High performing novice programmers are expected to have a lower regression ratio than their low performing counterparts. We calculated the regression ratio using Equation 9. We calculated the average regression ratio of participants in all five programs.

Eq. 9. Regression ratio = number of regressive saccades / total number of saccades in the stimuli

In Figure 1, the regression ratio is determined by dividing the number of regressive saccades, which in this case is 2 (D-E and E-F), by the total number of saccades made in response to the stimuli, resulting in a ratio of 0.40.

2.1.5 Statistical Analysis Conducted on the Saccade Data

To select suitable statistical tests for analysis, we conducted the Shapiro-Wilk test to assess

the normality of the data sets. Table 1 displays the results of the normal distribution tests conducted on the saccade metrics of compiler error messages, error lines, and entire stimuli, for both high and low performing participants.

Table 1. *Results of Normal Distribution Tests on Saccade Metrics*

AOI	Class	Average Saccade Ratio Result	Class	Average Saccade Rate Result
Compiler Error Messages	High	Not Normal	High	Not Normal
Compiler Error Messages	Low	Normal	Low	Not Normal
Error Lines	High	Normal	High	Not Normal
Error Lines	Low	Normal	Low	Not Normal
		Average Saccade Length		Average Regression Ratio
Entire Stimuli	High	Not Normal	High	Normal
Entire Stimuli	Low	Normal	Low	Not normal

We employed Welch's t-tests for normally distributed datasets and Mann-Whitney U tests for non-normally distributed datasets to determine whether significant differences exist in saccadic patterns of high and low performing novice programmers.

3. Results and Discussion

3.1 Difference of Saccadic Patterns to Compiler Error Messages of High and Low Performers

We observed no significant difference in the average saccade ratio on compiler error messages between high and low performing participants. Additionally, there was no significant difference in the average saccade rate on compiler error messages between the two groups of participants. The result of our analysis is summarized in Table 2.

Table 2. *Saccadic Patterns of High and Low Performing Novice Programmers (Compiler Error Messages)*

Saccade Metrics		High (N=27)		Low (N=18)		Mann-Whitney U test	
		M	*SD*	*M*	*SD*	*U*	*p*
Average Saccade Ratio		0.18	0.15	0.15	0.09	246.5	0.94
Average Saccade Rate		0.10	0.07	0.11	0.14	276.0	0.45

Note: $p < 0.05$ indicates statistical significance.

The lack of a significant difference in the average saccade ratio between high and low performing participants implies that both groups tend to exhibit similar patterns of eye movements when encountering compiler error messages. This could be an indication that both

high and low performing participants utilize similar strategies for quickly scanning code and identifying potential errors. Our finding is aligned with that of Uwano et al. (2006) that programmers during code reviews tended to initially scan through the entire lines of code briefly before focusing on specific portions. Also, the absence of a significant difference in the average saccade rate suggests that, despite differences in performance level, both groups were equally efficient in visual processing and navigating through code containing error messages and that regardless of performance level, novice programmers allocate comparable attention to the compiler error messages during debugging.

3.2 Difference of Saccadic Patterns to Error Lines of High and Low Performers

High performing participants did not exhibit a statistically significant difference in average saccade ratio on error lines compared to their low performing counterparts. Similarly, we found no significant difference in the average saccade rate on error lines of high and low performing participants. The findings of our analysis are outlined in Table 3.

Table 3. *Saccadic Patterns of High and Low Performing Novice Programmers (Error Lines)*

Saccade Metrics	High (N=27)		Low (N=18)		t-test	
	M	*SD*	*M*	*SD*	*t*	*p*
Average Saccade Ratio	0.18	0.11	0.17	0.12	0.21	0.83
	High (N=27)		Low (N=18)		Mann-Whitney U test	
	M	*SD*	*M*	*SD*	*U*	*p*
Average Saccade Rate	0.08	0.06	0.07	0.06	281.5	0.38

Note: $p < 0.05$ indicates statistical significance.

Our result on average saccade ratio implies that before fixation, both groups tend to exhibit similar scanning behaviors when encountering error lines. The lack of significant difference on average saccade rate on error lines could suggest that the ability to spot errors is not strongly correlated with saccade rate alone.

3.3 Difference of Saccade Lengths of High and Low Performers

We found that the high performing participants had significantly longer average saccade lengths compared to the low performing participants. Our findings support the notion proposed in the work of Busjahn et al. (2015) that experts can make larger jumps during reading to focus on important source code elements. This suggests that high performers may possess a similar ability to expert readers, allowing them to efficiently navigate through code by making larger and more purposeful eye movements to focus on significant elements in the program. Our analysis findings are shown in Table 4. In Figure 2, the distribution and skewness of average saccade length is illustrated. High performers exhibited a significantly higher median average saccade length compared to low performers.

Table 4. *Average Saccade Lengths of High and Low Performing Novice Programmers*

Saccade Metrics	High (N=27)		Low (N=18)		Mann-Whitney U test	
	M	*SD*	*M*	*SD*	*U*	*p*
Average Saccade Lengths	134.31	28.52	107.84	14.96	390.0	0.005*

Note: $p < 0.05$ indicates statistical significance.

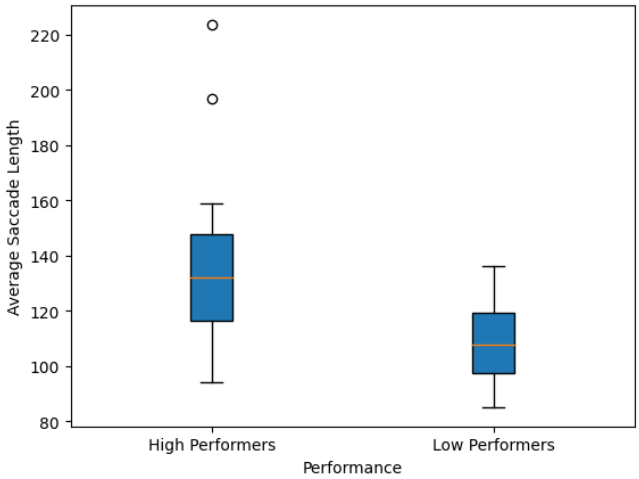

Figure 2. High and Low Performers' Average Saccade Length.

3.4 Difference of Regression Ratio of High and Low Performers

We found that high performing participants had significantly lower average regression ratio than their low-performing counterparts. Our findings support the conclusions drawn in previous studies, such as those by Goldberg and Kotval (1999) and Poole and Ball (2005), which reported that higher regression rates indicate increased difficulty in executing and completing a task. Bednarik and colleagues (2012) also discovered that repetitive gaze patterns, which may involve regressive saccades, in some context, were linked to lower levels of expertise among students debugging Java programs using a program visualization system. This suggests that high performers may encounter fewer challenges that require revisiting previously examined areas, leading to more efficient task execution and completion. We summarized the result of our analysis in Table 5. Figure 3 depicts the distribution and skewness of the average regression ratio. High performers demonstrated a significantly lower median average regression ratio in comparison to low performers.

Table 5. *Average Regression Ratio of High and Low Performing Novice Programmers*

Saccade Metrics	High (N=27)		Low (N=18)		Mann-Whitney U test	
	M	*SD*	*M*	*SD*	*U*	*p*
Average Regression Ratio	0.71	0.09	0.81	0.09	99.0	<.001*

Note: $p < 0.05$ indicates statistical significance.

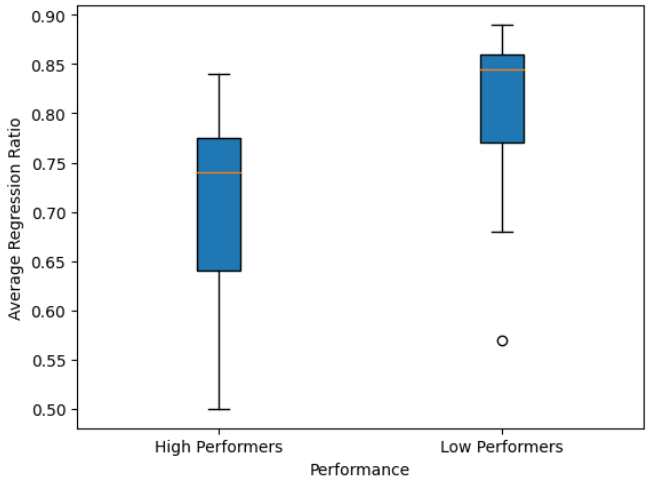

Figure 3. High and Low Performers' Average Regression Ratio.

4. Conclusion and Future Work

This study aimed to find insights into problem solving strategies employed by novice programmers using saccade-based metrics. Our findings suggest that the code reading strategies of the high and low performing participants may not be primarily driven by variations in initial scanning behaviors or the speed of eye movements when encountering compiler error messages. Instead, both high and low performing participants employ similar strategies for scanning through code and detecting potential errors. The longer average saccade lengths in high performing participants demonstrate a strategic navigation pattern wherein there is purposeful exploration of the code allowing high performers to gather more information in each eye movement. The lower average regression ratio in high performing participants suggests that they may demonstrate a greater ability to maintain focus and comprehension throughout the problem-solving task, resulting in fewer instances of regression.

Our findings in this study have some implications for developing teaching strategies in programming. The absence of significant differences in average saccade ratio and saccade rate suggests that teaching strategies should focus on promoting deep understanding, individualized support, and diverse instructional methods to enhance learning outcomes for all students. Educators may create interventions tailored to student needs, offering additional support to those with frequent regressive saccades, which may indicate difficulty in maintaining focus. Creating exercises focused on interpreting both literal and non-literal compiler error messages will help students become more comfortable with different types of errors. Teaching strategies should encourage students to adopt strategic code reading techniques, including lessons on systematically scanning and interpreting code, focusing on key elements first. Students can be taught techniques to reduce unnecessary regression while reading code, such as maintaining focus, minimizing backtracking, and breaking down complex problems into smaller and manageable parts.

A limitation of the study is the lack of integration of contextual factors such as task complexity, familiarity with programming languages, and individual learning styles into the analysis of saccade-based metrics. These factors could help identify situational influences on problem-solving strategies, and this will be considered in future work.

Acknowledgements

We thank Ateneo de Manila University and Pangasinan State University administration for

allowing us to conduct the experiment for this study.

References

Aljehane, S. (2022). Eye Movements Characterizing for the Assessment of Expertise in Source Code Reading, August 2022. http://rave.ohiolink.edu/etdc/view?acc_num=kent1658417479599309

Bednarik, R. (2012). “Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations,” *International Journal of Human-Computer Studies*, vol. 70, no. 2, pp. 143–155, Feb. 2012

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., & Tamm, S. (2015). Eye Movements in Code Reading: Relaxing the Linear Order. *2015 IEEE 23rd International Conference on Program Comprehension,* 255–265*.* https://doi.org/10.1109/ICPC.2015.36

Busjahn, T., Schulte, C., 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, ACM, New York, Koli Calling ’11*, pp 1–9. https://doi.org/10.1145/2094131.2094133

Dy, T. and Rodrigo, M. M. (2010). A detector for non-literal java errors. In Koli Calling ’10, Koli, Finland, October28–31, 2010. ACM.

Goldberg, J.H. and Kotval, X.P. (1999). Computer interface evaluation using eye movements: methods and constructs. *Int J Ind Ergon* 24(6):631–645

Goettker, A., Brenner, E., Gegenfurtner, K.R., de la Malla, C. (2019). Corrective saccades influence velocity judgments and interception. Sci Rep. 2019 Apr 1;9(1):5395. doi: 10.1038/s41598-019-41857-z. PMID: 30931972; PMCID: PMC6443687.

Holmqvist, K., Nyström, M., Andersson, R., Dewhurst, R., Jarodzka, H. and Van de Weijer, J. (2011). Eye tracking: A comprehensive guide to methods and measures. *OUP Oxford.*

Mahanama, B., Jayawardana, Y., Rengarajan, S., Jayawardena, G., Chukoskie, L., Snider, J. and Jayarathna, S. (2022). Eye Movement and Pupil Measures: A Review. *Frontiers in Computer Science. Sec. Human-Media Interaction* Volume 3 - 2021 available: https://doi.org/10.3389/fcomp.2021.733531

McChesney, I. and Bond, R. (2020). Observations on the Linear Order of Program Code Reading Patterns in Programmers with Dyslexia. *Association for Computing Machinery.* https://doi.org/10.1145/3383219.3383228

Ohtani, A. (1971). An Analysis of Eye Movements during a Visual Task. *Ergonomics* 14, 167–174. doi:10.1080/00140137108931235

Pacol, C., Rodrigo, M. M. and Tablatin, C. L. (2024). A Comparative Study of High and Low Performing Students’ Visual Effort and Attention when Identifying Syntax Errors. *In: Schmorrow, D.D., Fidopiastis, C.M. (eds) Augmented Cognition. HCII 2024. Lecture Notes in Computer Science(), vol 14694. Springer, Cham. https://doi.org/10.1007/978-3-031-61569-6_6*

Pacol, C., Rodrigo, M. M. and Tablatin, C. L. (in press). Exploring the Relationship of Personality Domains and Visual Attention Patterns in Novice Programmers. *International Conference on Computers in Education (ICCE 2024).*

Poole, A., & Ball, L. J. (2005). Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future. Prospects”, Chapter in C. Ghaoui (Ed.): Encyclopedia of Human-Computer Interaction. Pennsylvania: Idea Group, Inc.

Sharafi, Z., Sharif, B., Guéhéneuc, Y-G., Begel, A., Bednarik, R., and Crosby, M. (2020). A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering,* 25, 3128-3174

Tablatin, C. L. and Rodrigo, M. M. (2023). “Visual Attention Patterns in Processing Compiler Error Messages”, *Proceedings of the 31st International Conference on Computers in Education. Asia-Pacific Society for Computers in Education*

Uwano, H., Nakamura, M., Monden, A., and Matsumoto, K.-i. (2006). “Analyzing individual performance of source code review using reviewers’ eye movement,” *in Proc. of the Symposium on Eye Tracking Research & Applications*, San Diego, California: ACM, 2006, pp. 133–140