Yang, J. C. et al. (Eds.) (2018). Proceedings of the 26th International Conference on Computers in Education. Philippines: Asia-Pacific Society for Computers in Education

Characterizing Individual Gaze Patterns of Pair Programming Participants

Maureen VILLAMOR[a,b*] and Ma. Mercedes RODRIGO[a]
[a]*Ateneo de Manila University, Quezon City, Philippines*
[b]*University of Southeastern Philippines, Davao City, Philippines*
*maui@usep.edu.ph

Abstract: In this paper, we investigate the gaze patterns of individuals within pairs as they traced and debugged fragments of code. We performed a dual eye tracking experiment, recorded their fixations and computed the gaze-based metrics of these individuals. The participants within pairs were categorized into a more successful and less successful participant based on the number of bugs found. Results suggest that the more successful participants acquaint themselves first with the program encoding more information about the program, have more increased attention on the erroneous lines of code, strike a good balance between processing and searching, read code less linearly, and are more engaged in the task. The goal of this study is to capture individual expertise to gain insights on what makes the more skilled participants in a pair programming setup efficient and effective through their gaze patterns.

Keywords: Pair programming, eye tracking metrics, gaze patterns

1. Introduction

Pair programming is a collaborative work arrangement where two programmers execute different programming activities together. It may be co-located, i.e., programmers share a single screen or may also occur remotely or in spatially distributed mode in which programmers look at the same code but on different screens (Baheti et al., 2002). It has become a popular collaboration paradigm utilized in teaching introductory programming courses because it has been proven that pair programming is beneficial to students' learning and attitudes towards programming.

Recent publications related to programming have made use of eye gaze to assess the comprehension of C++ and Python source code (Turner et al, 2014), study how programmers read source code (Busjahn et al., 2015; Yenigalla et al., 2016; Jbarra & Feitelson, 2017), explore students' cognitive processes while debugging (Lin et al., 2016), observe differences on debugging strategies given diverse types of errors (Peng et al.,, 2016), and understand the learning process of coding with different age groups (Papavlasopoulou et al., 2017).

Because it is an indicator of attention, eye gaze has been linked to cognition (Just & Carpenter, 1976). This concept of visual attention, mapping it to cognitive states is based from the "eye-mind" hypothesis (Poole & Ball, 2006). Eye movements can be tracked to gain a better understanding of the path and focus of visual attention during a task. Cognitive processes involved in performing a specific task can be inferred if we know which objects have been visually inspected including their order and context. Because of this, eye tracking has become one of the promising tools to track down the cognitive processes in programming particularly program comprehension. Eye movements in terms of saccades (rapid movements) and fixations (short stops), for example, can reveal the details of cognitive processing and the allocation of visual attention within a programming task.

The goal of this paper is to investigate and characterize the individual gaze patterns in the context of pair programming particularly program tracing and debugging. Specifically, this paper attempts to answer the following question: What characterizes the individual gaze patterns of the more successful participants and the less successful participants in a pair programming setup? This study aims to investigate individual behavioral characteristics of the more successful and less successful participants to shed light on what makes one better than the other so that we can learn

from the practices of the more successful participants and be able to design remediations that can help those who are struggling in programming.

2. Novices vs. Experts on Program Comprehension and Debugging

Researchers confirm that there are indeed differences between experts and novices. Novices are characterized as having longer fixation duration and do not read code at the beginning; while experts have shorter fixation durations and spend significantly more time on initial code reading (Bednarik et al., 2006). Experts are found to perform better and have shorter scan times when reviewing source code compared to novices (Sharif, Falcone & Maletic, 2012). Novices exert more effort and experience more difficulty reading source code as they progress through the course (Yenigalla et al. 2016) and tend to adapt the linear "Story Order" when reading natural language text and source code, while experts read code less linearly than novices (Busjahn et al., 2015). Researchers speculate that novices' inability to comprehend programs is due to the novices' tendency to read the code in the order it appears rather than in the order it is executed (Nanja & Cook, 1987).

Programmers are categorized as novices and experts based on their ability to "chunk" the program they are debugging (Vessey, 1985). Novices exhibit more inconsistent debugging behavior due to their inability to chunk programs, and they use a depth-first search approach focusing on locating and correcting the error without understanding the overall program first. On the other hand, experts use a breadth-first search approach when debugging, acquainting themselves first with the program to obtain a systems view before trying to find the source of the bug. Lin et al. (2016) affirm these findings claiming that low-performers tend to debug programs in a trial-and-error manner, are more focused on syntax, and fail to create suitable mental models for debugging; while high-performers debug programs in a more logical manner because of their "chunking" ability and prior knowledge. Experts are likewise much better at debugging by comprehension while novices work in isolation to track down the errors one at a time (Nanja & Cook, 1987).

These studies were primarily focused on novice vs. expert categorization and their differences in an individual programming setup. This study is part of our pair programming eye tracking study where we endeavor to draw out the dissimilarities between individuals within programming pairs to understand their behaviors that lead to optimal error detection. We also want to find out whether "novice vs. expert" differences hold true for individuals within pairs.

3. Methods

3.1 Participants, Structure of the Study, Data Cleaning and Statistical Treatment

The study was conducted in 6 universities in the Philippines recruiting 2nd to 4th year level college students who had already taken their college-level fundamental programming course. Eighty-four (84) participants, 56 males and 28 females, were randomly paired regardless of gender, proficiency level, and acquaintanceship resulting in a total of 42 pairs. Two Gazepoint eye trackers were used to collect the pairs' eye movement data. The pairs were shown 12 programs with errors and were instructed to mark the location of the errors. There was no need to correct the errors. The participants were made aware how many bugs were there in each program. Each of the 12 programs either contained a syntax, logic, semantic or a combination of these types of errors. The programs were categorized as easy, moderate, or hard depending on the type of error the program contained.

A slide sorter program with "Previous", "Reset", "Clear/Finish" and "Next" buttons was created to display the program specifications followed by the erroneous programs. When a participant finds a bug, the location of the bug is marked using a mouse-click and the software automatically draws an oval on the location. It is possible to remove the oval by pressing the "Reset" button. The pairs were told to work with their partner on the problems and collaborate using a chat program. Although they were seated together in the same room, they were spaced far enough to ensure that all communications with their partner was via chat only. For a more detailed description on the structure of the study, see Villamor and Rodrigo (2018).

Results of the written program comprehension test, which was used to split the participants into high and low proficient participants, and the number of bugs identified were recorded. The slide

sorter program generated log files for every participant, which contained a recording of buttons pressed and if the participant has already marked the location of the bug, the timestamps when these buttons were pressed, the slide numbers of the program specification and actual programs, and the *x* and *y* screen coordinates of the ovals that appeared after the mouse click.

The fixation data was cleaned first by removing fixations with negative *x*- and *y*- gaze coordinates. The number of fixations per slide that contained the actual program were segmented and saved on separate files. Hence, each participant had at most 12 fixation files (some pairs did not finish the 12 programs). The segmentation was done with the help of the information contained in the slide sorter program log files. To test for statistically significant differences, a *t*-test for independent sample means at the 0.05 level of significance was performed.

3.2 Eye Tracking Metrics

Two primary eye movement measurements used in eye tracking research are fixations and saccades. Fixations refer to moments when the eyes are relatively stationary, which reveal that information from the scene is acquired. Saccades are quick eye movements occurring between fixations, which can be related to a searching sequence of particular areas of interest. We used fixation-derived and saccade-derived metrics to characterize the gaze patterns of the individuals within pairs. These metrics are the following: *fixation count overall and on-target, fixation rate overall, fixation duration mean, fixation/saccade ratio, average saccade length, complete fixation time on-target, time to first fixation on-target and regressive saccades overall and on-target.* To know about these metrics, refer to Poole and Ball (2006). The 12 programs served as the stimuli in this study. OGAMA software (Voßkühler, 2008) was used to calculate these metrics and to convert each erroneous line of code in the program to an area of interest (AOI) so that AOI-based metrics can be extracted specifically for these regions.

Other non-eye tracking parameters were also used such as the duration per program, the number of times a participant changed his/her answers through the "Reset" button from the slide sorter program, the number of switches from the program specification to the actual erroneous code, and the time spent reading the program specification. Aside from these said metrics, we were also interested in finding out who between the more successful and less successful participant within pairs had more incidences where they saw the bugs (had fixations on the AOI) and marked it, saw the bugs but did not mark it, and missed the bugs (no fixations on the AOI) but marked it. To do this, we used visualizations such as heatmaps and fixation maps. These visualizations can show the general distribution of fixations and attention spots to imply varying levels of attention.

4. Results and Discussion

Of the 84 participants from the 42 pairs, we recorded a total of 970 cases, where a case is defined as one of the programs under each pair. Each participant within pairs was categorized as a more successful participant (MSP) or a less successful participant (LSP) depending which participant found more or fewer bugs in the 12 programs combined. A median split could have been done for the entire sample to know where each participant was in terms of the larger sample and then see how many were in MSP/LSP pairings. However, this process would leave us with only 8 pairs and 192 cases to examine.

Two (2) pairs of participants had the same debugging scores, and hence, were excluded reducing the number of cases down to 927. Four of these 927 cases had fixation counts less than 10 and were also discarded. Of the remaining 923 cases, 464 of these were from the MSPs, while 459 of these cases were from the LSPs. The majority of the MSPs (72.62%) and LSPs (63.18%) are high and low proficient, respectively. Hence, it is safe to say that pair programming participants who are more likely to perform well in a bug-finding task are those that are highly proficient. In the discussion that follows, there is a tendency to associate MSPs with "experts" on the basis that the MSPs are more proficient in programming compared to the LSPs. To get the aggregate results, the eye movement data was averaged over each category (MSPs and LSPs). The mean and standard deviation of each significant metric for the two categories as well as the *t*-test results are shown in Table 1.

Table 1

Descriptive Values and T-Test Results of Each Significant Metric

Metric	MSP		LSP		*t*-value	*p*-value
	Mean	Std. Dev.	Mean	Std. Dev.		
Fixation Count O/A	545	420	491	369	2.057	0.040
Fixation Count O/T	72	69	61	64	2.490	0.013
Fixation Rate O/A	1.15	1.03	1.35	1.07	-2.816	0.005
Fix./Saccade Ratio	322.59	278.97	387.18	295.98	-3.412	0.001
Ave. Saccade Length	124.32 px	28.33 px	129.57 px	31.06 px	-2.684	0.007
Regressions O/A	123.74	96.58	111.17	85.35	2.096	0.036
Comp. Fix. Time O/T	21.32 s	20.06 s	18.15 s	19.87 s	2.411	0.016
Duration Per Program	1105.62 s	950.13 s	899.92 s	862.44 s	3.443	0.001
Saw & Marked the Bug	1.58	0.94	1.32	0.88	4.398	0.000
Saw and Did Not Mark	0.88	0.92	1.02	0.96	-2.261	0.024
Missed but Marked	0.00	0.00	0.04	0.22	-3.918	0.000

The MSPs had significantly higher fixation count overall and on-target compared to the LSPs. This is contrary to previous research findings that highly proficient participants have lower fixation counts than the low proficiency participants. Several possible explanations could account for this result. First, since this was designed primarily as a pair programming task, the results could be different from an individual programming task where the participants work independently and are categorized as novices and experts. Second, this is a pair programming task where the participants were encouraged to collaborate and communicate with partners. Hence, it is possible that the fixations that occur while they were chatting might increase the fixation counts of the MSPs. Third, this is in line with prior research finding that experts acquaint themselves first with the program before trying to find the source of the bug, thus the higher fixation counts; while novices just focus on bug finding without comprehending first the overall program (Vessey, 1985).

Fourth, it is possible that the MSPs tend to refixate on the words in the text before leaving the word, which are often caused by originally landing in a "bad" place in a word and that processing of the word is distributed over two or more fixations in (O'Regan & Lévy-Schoen, 1987). Fifth, the MSPs higher fixation count on-target is an indication that the errors are more easily noticed by the MSPs and they have more fixations on the bugs possibly to analyze how the bugs affect the program. Lastly, the LSPs lower fixation counts could mean that they are less engaged in the task, could just be waiting for their stronger partners to give the answer, and have less attention to detail not realizing that those are the bugs. Upon further inspection, it was found that the MSPs higher fixation count overall and on-target were only significant on hard programs. This implies that the MSPs, compared to LSPs, are encoding more information about the program particularly the hard ones. The MSPs higher fixation count is not a result of searching less efficiently as prior literature would suggest.

Even if the fixation counts overall and on-target of the MSPs are higher than the LSPs, it seemed unexpected that the MSPs fixation rate overall was significantly slightly lower than the LSPs. From prior literature, this is an indication that information processing may not be effective. However, this may not be true because the MSPs were more successful in finding bugs. The MSPs had more fixations overall but the rate at which they fixate on the program elements was not high enough to deter them from doing enough searching to try to find all the bugs. The LSPs higher fixation rate overall and on-target could mean that they are just randomly sampling lines of codes, which implies trial and error in locating bugs.

The MSPs had significantly lower fixation/saccade ratio than the LSPs. Since this metric can be used as an index of processing vs. searching (Goldberg & Kotval, 1999), it is possible that the MSPs have spent less time processing and more searching because the task required them to find as many bugs as possible within the time allotted. However, the MSPs had more fixations on-target (indicative of more processing) and had shorter saccade lengths (indicative of less searching). This could signify that the MSPs possibly employ a balance mix of processing and searching or there is a

greater diversity of gaze patterns among the MSPs that reflect their individual case-based knowledge (Robertson, 2016).

The MSPs had significantly shorter average saccade lengths than the LSPs, which is contrary to previous findings that experts are more likely to make longer saccades than novices. One possible explanation is that this could be influenced by the way the text in the source code were formatted, particularly longer programs, to fit the entire screen (no scrolling needed) so the distances among bug locations do not require longer jumps. Since the lines were condensed to each other, this might have affected the saccade lengths of the MSPs. The LSPs longer average saccade lengths may be an indication that they might be searching randomly to attempt to locate the bugs. From prior literature, long saccades are less accurate and lead to ill-placed fixations (Rayner et al., 2012).

The regressive saccades overall of the MSPs was significantly more frequent than the LSPs but the difference in their regressive saccades on-target was not significant. Prior research suggest that more incidences of regressive saccades is an indication of processing difficulty, confusion, or problems in understanding (Rayner et al., 2012). However, it could be argued that this is probably a form of "good confusion" since the MSPs found more bugs, and hence, were more productive. It is also possible that this could be a result of following the code execution order (e.g., tracing a *for* loop) or an indication that the MSPs read code less linearly than LSPs (Busjahn et al., 2015). Prior research also claim that many regressions are due to comprehension failures, that is, when readers encounter a word which they have misinterpreted, they often make regressions as soon as they encounter disambiguating information. These regressions typically are very short saccades and are probably due to oculomotor errors (Rayner et al., 2012). This could also explain the MSPs more frequent regressions overall.

The MSPs complete fixation time on-target was significantly longer than the LSPs. However, this significance holds true only for the moderate and hard programs. This suggests that the MSPs could be taking their time processing and could be exhibiting a higher level of interest on each error found particularly when program difficulty increases. The difference on the duration per program between the MSPs and LSPs was also significant. The MSPs longer time spent in each program could be an indication that they are more engaged and are willing to take an extra step to locate all the bugs in the programs compared to the LSPs who are more likely to disengage from the task especially if the program complexity increases and tend to hastily start solving the problems without analyzing them.

The MSPs had significantly more incidences where they fixated on the bugs and marked it, which matched our expectations. The difference, however, in comparison to the LSPs was not quite high, which could mean that there could be instances where the MSPs have told the LSPs the answers. It did not come as a surprise that the LSPs had significantly more occasions where they came across the errors but did not mark it. This means that the LSPs were not aware that those were the erroneous line(s) of code. Lastly, none of the MSPs were able to miss the bug(s) but marked it but the LSPs had few instances where they did not fixate on the bug(s) and yet they were able to mark it correctly, which implies that the MSPs had found the bug(s) and told the LSPs.

5. Summary, Implication, and Future Work

This paper investigated the individual gaze patterns of the more successful and less successful participants within programming pairs to help us understand what makes the other succeed in a bug-finding task. In summary, the MSPs are characterized as having higher fixation count overall and on-target but lower fixation rate overall, lower fixation/saccade ratio, shorter average saccade length, longer complete fixation time on-target, more frequent regressions overall, and longer duration per program. The LSPs, on the other hand, have the exact opposite characterizations. These are indications that programming pair participants aside from their usual assigned roles as "driver" and "navigator" can also be distinguished based on their individual gaze patterns.

Programming is not an easy task and because of its challenging nature, programming educators strategize to make programming less difficult and one of these strategies is to implement pair programming. The significance of this study is to have a better understanding on the behaviors of the individuals within programming pairs through their gaze patterns while they trace and find bugs. Programming professors can learn from the behaviors of the more successful individuals within pairs.

Since these are just initial findings on the behaviors of the pair programming participants, the next step would be to conduct this experiment further to assess how these behaviors impact the performance of the individuals within pairs as well as their pair performance. We wish also to find out which program elements do MSPs and LSPs primarily inspect so that we can get a clearer picture on their respective cognitive processes. We will also look at the pairs' behavior when they collaborate, assess the pairs' collaboration based on the composition of the pairs, and examine their impact on the success of the pairs. The end goal of this research is to be able to create an explanatory model that could explain the dynamics that take place in a pair programming setup.

References

Baheti, P. (2002, November). Assessing distributed pair programming. In *Companion of the 17th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications* (pp. 50–51). ACM.

Bednarik, R., Myller, N., Sutinen, E., & Tukiainen, M. (2006). Analyzing individual differences in program comprehension. *Technology Instruction Cognition and Learning*, *3*(3/4), 205.

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., ... & Tamm, S. (2015, May). Eye movements in code reading: Relaxing the linear order. In *Program Comprehension (ICPC), 2015 IEEE 23rd International Conference on* (pp. 255–265). IEEE.

Goldberg, J. H., & Kotval, X. P. (1999). Computer interface evaluation using eye movements: methods and constructs. *International Journal of Industrial Ergonomics*, *24*(6), 631–645.

Jbara, A., & Feitelson, D. G. (2017). How programmers read regular code: a controlled experiment using eye tracking. *Empirical software engineering*, *22*(3), 1440–1477.

Just, M. A., & Carpenter, P. A. (1976). Eye fixations and cognitive processes. *Cognitive psychology*, *8*(4), 441–480.

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.

Nanja, M., & Cook, C. R. (1987, December). An analysis of the on-line debugging process. In *Empirical studies of programmers: second workshop* (pp. 172–184). Ablex Publishing Corp..

O'Regan, J. K., & Lévy-Schoen, A. (1987). Eye-movement strategy and tactics in word recognition and reading. In M. Coltheart (Ed.), *Attention and performance: Vol. 12. The psychology of reading* (pp. 363–383). Hillsdale, NJ: Erlbaum.

Papavlasopoulou, S., Sharma, K., Giannakos, M., & Jaccheri, L. (2017, June). Using Eye-Tracking to Unveil Differences Between Kids and Teens in Coding Activities. In *Proceedings of the 2017 Conference on Interaction Design and Children* (pp. 171–181). ACM.

Peng, F., Li, C., Song, X., Hu, W., & Feng, G. (2016, June). An Eye Tracking Research on Debugging Strategies towards Different Types of Bugs. In *Computer Software and Applications Conference (COMPSAC), 2016 IEEE 40th Annual* (Vol. 2, pp. 130–134). IEEE.

Poole, A., & Ball, L. J. (2006). Eye tracking in HCI and usability research. *Encyclopedia of human computer interaction*, *1*, 211–219.

Rayner, K., Pollatsek, A., Ashby, J., & Clifton Jr, C. (2012). *Psychology of reading*. Psychology Press.

Robertson, S. I. (2016). *Problem solving: perspectives from cognition and neuroscience*. Psychology Press.

Sharif, B., Falcone, M., & Maletic, J. I. (2012, March). 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* (pp. 381–384). ACM.

Turner, R., Falcone, M., Sharif, B., & Lazar, A. (2014, March). An eye-tracking study assessing the comprehension of C++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (pp. 231–234). ACM.

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. *International Journal of Man-Machine Studies*, *23*(5), 459–494.

Villamor, M. and Rodrigo, M. M. (2018). Predicting Successful Collaboration in a Pair Programming Eye Tracking Experiment. In *Adjunct Publication of the 26th Conference on User Modeling, Adaptation and Personalization*. ACM.

Voßkühler, A. D. R. I. A. N., Nordmeier, V., Kuchinke, L., & Jacobs, A. M. (2008). OGAMA (Open Gaze and Mouse Analyzer): open-source software designed to analyze eye and mouse movements in slideshow study designs. *Behavior research methods*, *40*(4), 1150–1162.

Yenigalla, L., Sinha, V., Sharif, B., & Crosby, M. (2016, July). How novices read source code in introductory courses on programming: An eye-tracking experiment. In *International Conference on Augmented Cognition* (pp. 120–131). Springer, Cham.