How Novices Read Source Code in Introductory Courses on Programming: An Eye-Tracking Experiment

Leelakrishna Yenigalla[1], Vinayak Sinha[1], Bonita Sharif[1(✉)], and Martha Crosby[2]

[1] Youngstown State University, Youngstown, OH 44555, USA
{lkyenigalla, vsinha}@student.ysu.edu, bsharif@ysu.edu
[2] University of Hawaii at Mānoa, Honolulu, HI 96822, USA
crosby@hawaii.edu

Abstract. We present an empirical study using eye tracking equipment to understand how novices read source code in the context of two introductory programming classes. Our main goal is to begin to understand how novices read source code and to determine if we see any improvement in program comprehension as the course progresses. The results indicate that novices put in more effort and had more difficulty reading source code as they progress through the course. However, they are able to partially comprehend code at a later point in the course. The relationship between fixation counts and durations is linear but shows more clusters later in the course, indicating groups of students that learned at the same pace. The results also show that we did not see any significant shift in learning (indicated by the eye tracking metrics) during the course, indicating that there might be more than one course that needs to be taken over the course of a few years to realize the significance of the shift. We call for more studies over a student's undergraduate years to further learn about this shift.

Keywords: Eye tracking study · Program comprehension · Novices

1 Introduction

Programming involves both reading and writing source code [1]. Present computing education focuses on teaching how to write code, by taking reading skills for granted. Code reading which is also an important part of program comprehension is rarely considered [2]. If we understand how novices read source code, and what difficulty they face during initial learning, we can design better instruction tools, and educational environments. In this paper, we present an empirical study using eye tracking equipment to understand how novices read source code in the context of two introductory programming classes. Our main goal is to begin to understand how novices read code and determine any improvements in program comprehension as the course progresses. We use an eye tracker in the study as it is known that visual attention triggers mental processes in order to comprehend and solve a given task and effort put visually is directly linked to the cognitive effort [3]. A fixation is when the eye stabilizes on a

D.D. Schmorrow and C.M. Fidopiastis (Eds.): AC 2016, Part II, LNAI 9744, pp. 120–131, 2016.
DOI: 10.1007/978-3-319-39952-2_13

particular location for a particular duration. Saccades are quick movements between eye fixations. A *scan path* is a directed path formed by saccades between fixations. According to eye tracking literature, processing of visual information occurs during fixations but no processing occurs during saccades [4, 5].

The main contribution of this paper is an eye tracking empirical study that assesses how students, in particular novices read source code during a semester. Data collection was done using two methods: online questionnaires and an eye tracker (hardware and software). The ultimate goal is to develop better teaching strategies specifically targeted to novices and how they learn. To achieve this long-term goal, we first need to conduct several studies to determine individual behavior and determine if any differences exist before we can generalize this process. In this paper, we seek to answer the following research questions.

- RQ1: What progress do novices make as they go through a programming course?
- RQ2: Can we determine the difference in novice accuracy and progress using eye tracking data?
- RQ3: What are the similarities and differences in eye gaze between different tasks as time progresses in a course setting?

The paper is organized as follows. We give a brief description of related work in Sect. 2 followed by details on the experimental setup in Sect. 3. Section 4 presents observations and results. We present a discussion of the results in Sect. 5 followed by conclusions with ideas for future work.

2 Related Work

Busjahn and Schulte [1] conduct a study to find the importance of code reading and comprehension in teaching programming. They interviewed instructors to determine the importance of code reading in five categories: conceptualization, occurrences, and effects of successful code reading, challenges for learners, as well as approaches to facilitate code reading. The results tend to show that code reading is connected to comprehending programs and algorithms.

Sharif et al. study the impact of identifier style (i.e., camel case or underscore) on code reading and comprehension using an eye-tracker [6, 7]. They find camel case to be an overall better choice for comprehension. Sharafi et al. [8] extended this work and conducted an eye tracking study to determine if gender impacts the effort, time, and ability to recall identifiers.

Turner et al. [9] conducted a comparison study between C++ and Python source code to assess effect of programming language on student comprehension. They found no statistical difference between C++ and Python with respect to accuracy and time, but there is a significant difference between C++ and Python for fixation rate on buggy lines of code for find bug tasks.

Crosby et al. studied the effect of beacons on program comprehension and stated that beacons may be in the eye of the beholder [10]. Fan noted similar results on a study done in 2010 affirming results that suggest beacon identification is in the eyes of the beholder but noticed that the presence of comments does have an effect on code reading [11].

Hansen et al. looked into factors that made code hard to understand [12]. It was conducted online and used variants of Python source code to determine difficulty. The study was conducted on a wide range of subjects with varied programming experience. Their results led to a better understanding that experience helps when programmers believe that there may be errors but can actually hinder their ability when they haven't been trained for specific cases. Their results also showed that vertical whitespace was a factor in grouping sections of related statements together.

Busjahn et al. [13] talk about the relevance of eye tracking in computer education. In more recent work, Busjahn et al. [14] look into how source code reading differs from reading natural language and find that there is indeed a significant statistical difference.

3 The Study

An overview of the experiment is shown in Table 1. In order to understand the progression of the novice's understanding, we decided to conduct the experiment in two phases. The first phase was held in September 2014 and the second phase was held in November 2014. Different material was covered in class before each phase was conducted. The novice was instructed to read the code snippets shown to them and answer a comprehension question. They were also asked about the level of difficulty and their confidence level of answering the questions related to the code snippets.

The main dependent variables we want to determine that might be affected by the two phases are the accuracy, time, fixation count, and fixation duration.

Table 1. Experiment overview

Goal	To understand how novices read source code
Main factor	Time between testing: Phase 1 and Phase 2
Dependent variables	Accuracy, time, fixation count, fixation duration
Secondary factor	Class (Group1, Group2)

3.1 Hypotheses

Based on the research questions presented above, four detailed null hypotheses on each of the four dependent variables are given below.

H_a: There is no significant difference in accuracy between Phase 1 and Phase 2.
H_t: There is no significant difference in time between Phase 1 and Phase 2.
H_{fc}: There is no significant difference in fixation counts between Phase 1 and Phase 2.
H_{fd}: There is no significant difference in fixation durations between Phase 1 and Phase 2.

Alternative Hypotheses: There is a significant difference in accuracy, time, fixation count, and fixation duration when it comes to the two phases. Thus it is expected that if some improvement in learning occurs then there will be large differences between Phase 1 and Phase 2.

3.2 Participants

We recruited students from two classes in Fall 2014 at Youngstown State University (YSU). The first class was an object oriented beginner programming class and the second class was the server-side class on web development. We refer to the OOP class as Group 1 and the Server side class as Group 2 throughout the rest of the paper. There were 11 and 10 students in the OOP class and the server side class respectively. The syllabus for the OOP class was observed during the creation of the tasks. We asked our participants to self-assess their skills in a pre-questionnaire. Even though Group 2 had students with slightly more experience than Group 1, they were still novices. Figure 1 shows demographics of participants.

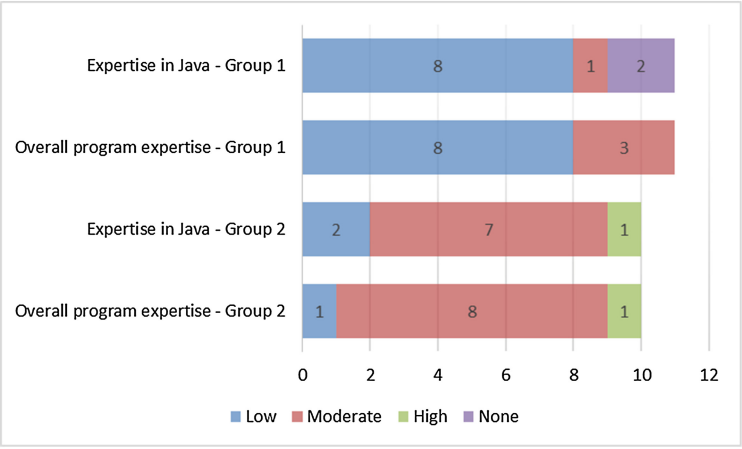

Fig. 1. Java expertise in both groups (Color figure online)

3.3 Tasks

We designed the tasks for both the phases of the study based on what the novices covered in the class syllabus for OOP in Group 1. In Phase 1, the tasks involved the following topics: array of strings, static keyword, random, substring, and static class methods. In Phase 2, the tasks involved topics such as GUI and events, exception handling, validation and OO inheritance and polymorphism. See Table 2 for an overview of the tasks used in the study. The complete set of study questions and programs including all background questions and post questionnaires can be found at http://seresl.csis.ysu.edu/HCII2016.

This study had six stimuli (programs) in each phase. The first phase had two tasks: "What is the output?" and "Give a summary". In the second phase we restricted the task to only giving a summary. There were easy, medium, and difficult programs with varying length to test different scenarios. In Phase 2, a program that was conceptually similar to a program is Phase 1 was also used to see if it was easier to comprehend after a couple of months. The *Primes* program was compared with *CheckString* from phase 2

in the difficult category, *StringProcessingDemo* was compared with *TextClass* and *Count* was compared with *PrintPattern*. They are shown in bold in Table 2.

Table 2. Overview of tasks and programs used in the study

Program name	Task	Overview of the program	Difficulty level	Phase 1	Phase 2
StringCheck	Output	String comparison	Easy	X	
Primes	**Summary**	**Finds all primes <=n**	**Difficult**	**X**	
TestPassArray	Output	Swapping array elements	Medium	X	
StringProcessingDemo	**Summary**	**Change part of a string**	**Easy**	**X**	
Rectangle	Output	Area of a rectangle	Difficult	X	
Count	**Summary**	**Number of times a letter occur in a string**	**Medium**	**X**	
TextClass	**Summary**	**Change part of a string**	**Easy**		**X**
TestingCircle	Summary	Exception handling	Medium		X
CheckString	**Summary**	**Check if string input is a palindrome**	**Difficult**		**X**
DoSomething	Summary	Selection sort	Easy		X
PrintPattern	**Summary**	**Prints three rows of stars in triangle**	**Medium**		**X**
KeyboardPanel	Summary	Draws a letter and moves it using arrow keys	Difficult		X

3.4 Data Collection and Apparatus

All subjects answered the six tasks in each phase via an online questionnaire presented as a Google Form. Each question was timed and the subjects' eyes were tracked. The subjects had to type the answer in the space provided in the online forms after they finished each task. We obtained IRB training and approval before we began this study.

The Tobii X60 eye tracker was used in this study. It is a 60 Hz video-based binocular remote eye tracker that does not require the user to wear any head gear. It generates 60 samples of eye data per second. The average accuracy for the Tobii eye

tracker is 0.5 degrees which averages to about 15 pixels. The eye tracker compensates for head movement during the study. The study was conducted on a 24 in. monitor with screen resolution set at 1920 * 1080. The eye gaze data includes timestamps, gaze positions, fixations and their durations, pupil sizes, and validity codes. In this study, we only analyze fixations and their durations.

4 Study Results

We now present the results and seek to provide answers to our research questions posed in the Introduction. Since our data is not normal and we have a small sample size, we use non-parametric measures to determine significance using the Wilcoxon paired test with alpha set at 0.05 that determines significance with a 5 % error and 95 % confidence.

4.1 Accuracy and Time

Each of the twelve programs were anonymously scored by the first author as fully correct, partially correct, or completely incorrect where partially correct got a rating of 0.5 and fully correct got a score of 1. With respect to correctness in Group 1, Wilcoxon test shows that Phase 1 total accuracy was significantly less accurate than Phase 2

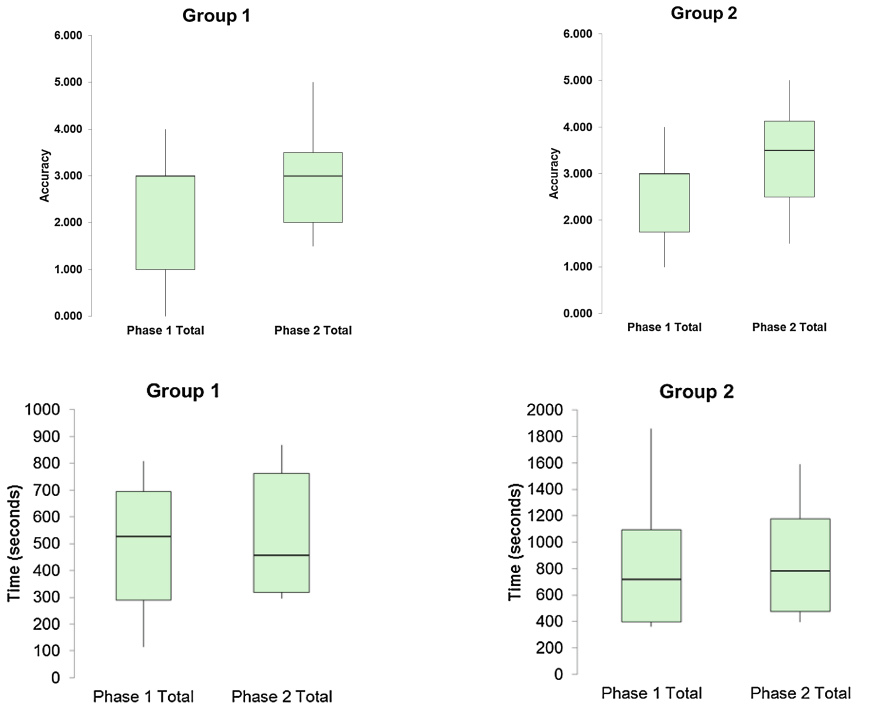

Fig. 2. Results for accuracy and time – Group 1 and Group 2 (both phases)

($p = 0.045$). In Group 2, Wilcoxon test also shows that Phase 1 total accuracy was significant less than Phase 2 ($p = 0.010$). We see that in Group 1, novices were more accurate in Phase 2 compared to Phase 1. They were also able to partially give answers to the programs in the second phase. Refer to Fig. 2 for the descriptive statistics.

With respect to time, there was no significant difference between Phase 1 and Phase 2 in terms of overall time of both groups. However we did find a significant difference between the following programs (Refer to Table 2 for more information on the tasks).

- *Count* and *PrintPattern* ($p = 0.012$) – Medium difficulty
- *StringProcessingDemo* and TextClass ($p = 0.049$) – Easy difficulty

where the program task in Phase 2 took longer to solve.

We notice that the *StringCheck* program that was the easiest was done in the least amount of time with not much variation between the subjects. The *Primes* and the *Count* programs were at the difficult and medium level of difficulty respectively. We notice that there is a much larger variation among the students for the harder programs that involve more programming constructs. In Phase 2, we found that the easy program *TextClass* took subjects longer to solve than its comparable counterpart in Phase 1 (i.e. *StringCheck*). We could speculate that this is because the novices start to focus more at the code and genuinely try to understand it. We notice this similar trend for Group 2 for the easy programs (i.e. *StringCheck* and *TextClass*). Group 2 took longer for almost all the tasks since they were more experienced than Group 1 and put in the effort to understand the programs to produce a reasonably correct answer. We also wanted to determine if there is any correlation between time and accuracy. We find that with increased amount of time, we see higher accuracy. In Phase 2, we again notice a step shift for two sets of students indicating that they might have learned in a similar fashion throughout the course.

We also compare programs that are similar in concept in Phase 1 and Phase 2 to determine if learning improved. The program *Count* is matched with *PrintPattern*, *Primes* is matched with *CheckString*, and *StringProcessingDemo* is matched with *TextClass* in Phase 1 and Phase 2 respectively.

Group 1 - We find that in the case of *Count* and *PrintPattern*, many of the students could not get the program correct in Phase 1 but many of them got *PrintPattern* partially correct in Phase 2. Comparison between *Primes* and *CheckString* shows that half of the total students of Group 1 showed partial improvement in accuracy solving *Checkstring* in Phase 2, hence there is no big difference in accuracy. When we compare *StringProcessingDemo* with *Textclass*, we noticed that three novices answered better in the second phase and four novices answered with the same level of accuracy (correct) in both phases. In terms of time, novices spent more time on the *Count* program in Phase 1. For *Primes* and *CheckString*, this was reversed because a majority of the students spent more time on *CheckString* from Phase 2. For the third set of programs we find that students spent more time reading *StringProcessingDemo* of Phase 1 than *TextClass* of Phase 2.

Group 2 - Between *Count* and *PrintPattern*, we found a huge jump in accuracy for *PrintPattern* in Phase 2. For the second set, four students had the same level of accuracy. Two of them answered better in the second phase with two answering incorrectly in the second phase. In terms of time, we see that half of the students spent

more time reading *Count* of Phase 1 compared to *PrintPattern* of Phase 2. For *Primes* vs. *CheckString*, four students took longer for *Primes* (Phase 1) than *CheckString* in Phase 2. Six students took longer for the *StringProcessingDemo* in Phase 1 than the program in Phase 2. This could indicate higher cognitive load.

4.2 Fixation Counts and Fixation Durations

Before we calculate fixation counts and durations we need to define areas of interest (AOIs) for which to collect them in. In this study, an AOI comprises each line of source code. The number of AOIs are equivalent to the number of lines in the program. We do not count any eye gazes that fall outside these lines i.e., blank space. In this paper, we only focus on the total lines AOI and not on an individual line-level analysis. The Wilcoxon test did not report any significant results for fixation counts or fixation durations between phases. In Phase 1, the highest fixation counts were on *Primes*, and the *Count* program. In Phase 2, *TestingCircle* and the *DoSomething* programs were the hardest for the participants. Only one participant got the *DoSomething* program correct in Group 1. None of the students got the *DoSomething* program correct in Group 2. The same trend is observed in the fixation counts in Group 2 for Phase 1 and Phase 2. See Fig. 3 for fixation counts in Group 1 for both phases.

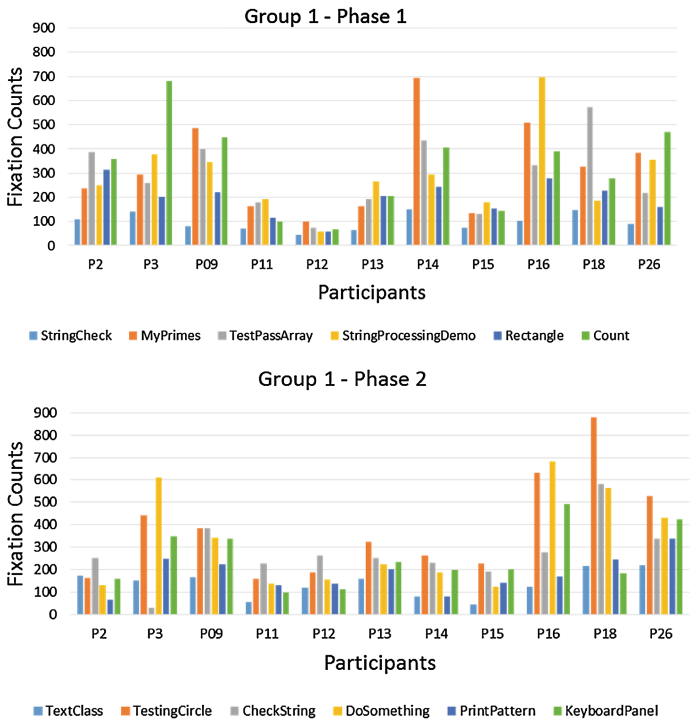

Fig. 3. Fixation counts for Group 1 (both phases) (Color figure online)

See Fig. 4 for fixation durations for Group 1. We have provided the fixation duration figures for Group 2 on the supplementary website listed in Sect. 3.3. In Group 1, the fixation durations were higher for *Primes* and *Count* programs in Phase 1. In Phase 2, *TestingCircle* and *DoSomething* had the highest fixation durations and the most variability between subjects. This indicates that not all the students were learning the concepts at the same rate. The standard deviation of the distribution for these programs was larger than the others.

In Group 2, for Phase 1, we find the most fixation durations on the *Primes* and *StringProcessingDemo* programs. In Phase 2, *TestingCircle* was the most difficult program since it had the highest number of mean fixation durations. This indicates higher cognitive load which is clearly harder for the novices to understand. Regressions (fixating over the same lines over and over) is also noticed for these programs.

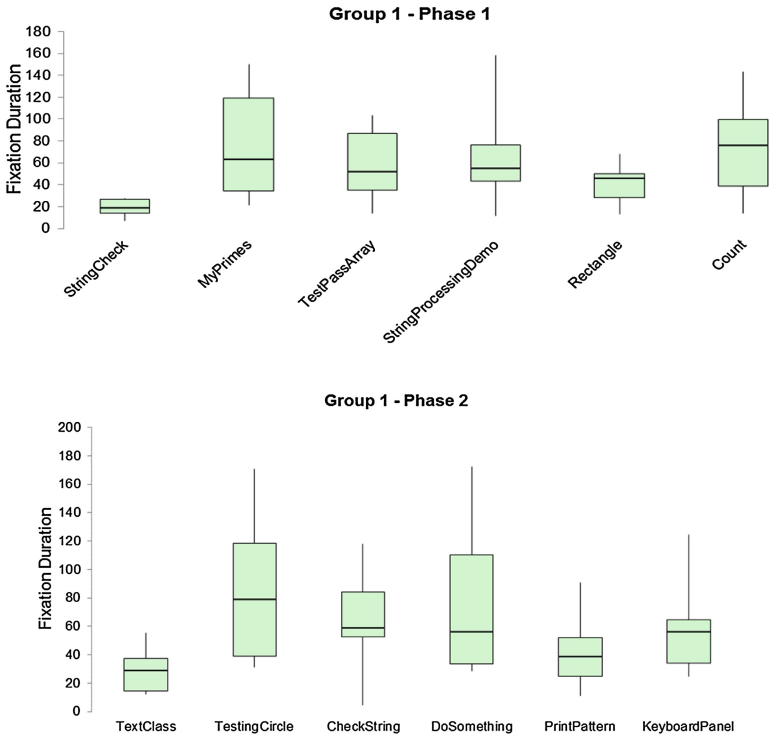

Fig. 4. Fixation durations for Group 1 – Phase 1 and Phase 2

We now discuss any relationship, if one exists, between the two dependent variables: fixation count and fixation duration. We notice that as the fixation count increases the fixation duration also increases in a linear fashion. In Phase 2, the relationship is more clustered indicating groups of students that learned at the same pace. We can clearly see three clusters in Fig. 5. The slope of both these graphs (Phase 1 (not shown) and Phase 2) is not the same indicating that in Phase 2 novices took the study more seriously taking longer time to accurately answer the questions.

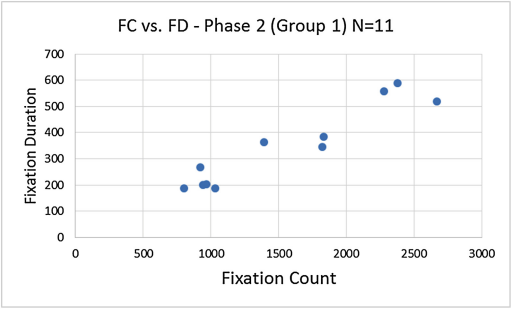

Fig. 5. Fixation count vs. fixation duration for Group 1 – Phase 2

4.3 Threats to Validity

We discuss how we minimize the main threats to validity in our study. The research participants did not know about the hypotheses used in the research. They only knew that they would participate in helping us understand how code is read and summarized. During the study, there was minimal contact between the experimenter and the participants. The experimenter did not interact or direct the participants to complete the questions in one way or another. We used the Wilcoxon paired test since we were comparing the same subject across Phase 1 and Phase 2. We used non-parametric measures due to our low sample size and non-normality of the data.

We tested this in only two classes at YSU. More tests need to be done to generalize the findings presented here. It is possible that there could have been some syllabus deviations in the classes that was unknown to experimenters which might have caused comprehension problems, however, after interviewing the students, this did not seem to be the case. They were exposed to all the ideas tested. Group 2 students were made aware of the topics that were covered in the object oriented class (Group 1).

In the first phase, we asked the subjects two types of tasks: determine the output and summarize the program. In the second phase, we only asked the subjects to summarize the programs. This could affect the results as well. We found that the summary included the output as a subset i.e., most students state the output of the program in the summary.

5 Discussion

We did not observe much progress in the novices in Group 1 however, there was a tendency to partially understand what the program is about. It is possible that one course is not enough to have a student master the concepts in Java. Ideally, we would want to follow a novice until they graduate to determine when the change in their mental model occurs. In one of the programs with a ternary operator, a novice was having a hard time understanding it as evidenced by many fixations and regressions on that line. We also noticed that novices tend to read source code like reading natural language text as pointed out in [14]. They do it in a linear sequential fashion.

We now revisit our hypotheses as presented in Sect. 3.1. Based on the results presented in Sect. 4, we are able to reject H_a as we do see higher accuracy in Phase 2 tasks. With respect to H_t when we take into account all the tasks together and compare Phase 1 and Phase 2 as a whole, we are not able to reject this hypothesis. We also did not find any significant differences between fixation counts and durations between Phase 1 and Phase 2. We are unable to reject H_{fc} and H_{fd}. Some possible explanations could be that the differences are more fine-grained and much more individualized to the specific student. A more fine-grained analysis across and between lines in the programs might produce a different result. We have left this as a future exercise.

6 Conclusions and Future Work

How does a novice read and comprehend code? In order to start to answer this question, we conducted a semester long experiment to determine if students learn the concepts presented during the semester. We ran the study in two phases with a 7-week separation between phases. We hypothesize that the students will be more accurate and spend less time on programs that they are familiar with when compared with other new and unfamiliar ones. We found that in both groups (classes) the novices were significantly less accurate in Phase 1 than Phase 2 but there was no significant difference in terms of time. In fact, more time was spent in Phase 2 to produce a correct answer. This came as a surprise at first but on second thought, we believe students were trying hard to understand code and put in more effort even though they clearly found the tasks difficult. Group 2 (still novices) had a slightly higher expertise in Java performed slightly better but not significantly. Also, in Phase 2 they were partially able to tell what the program did whereas in Phase 1 they bluntly stated that they did not know what the program's output was or how to summarize the program.

In the future, we plan on conducting a line by line analysis of the programs to identify patterns of lines that the novices looked at most and least. This will give us some idea on which parts of the program they looked at the most and had the hardest time with. A look at the transitions between beacons and chunks in the programs will also be beneficial since it has been shown that experts tend to chunk things together. For example, many of the novices did not realize one of the programs shown in Phase 2 was a sorting program. They were not able to chunk yet, however if we analyze their line-level transitions, it might provide a better indication of how they comprehend the loops involved in the sort. We also plan to continue this research by conducting this study at a much lower level such as CS0 or CS1 so we have a different aspect of how students learn when they are exposed to no programming language whatsoever. Finally, we believe one way to determine when a novice student is on their way to becoming an expert (i.e., mastering concepts), is to follow a small sample of students across their undergraduate study sampling eye movements on relevant tasks. Such a study will help us study progression as it occurs and then perhaps we could say that a shift in a student's mental model has occurred.

Acknowledgements. We thank all the students for their time and participation in this study. Many thanks to Teresa Busjahn for providing inspiration for some tasks in the study.

References

1. Busjahn, T., Schulte, C.: The use of code reading in teaching programming. In: Proceedings of the 13th Koli Calling International Conference on Computing Education Research, New York, NY, USA, pp. 3–11 (2013)
2. Busjahn, T., Schulte, C., Busjahn, A.: Analysis of code reading to gain more insight in program comprehension. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research, New York, NY, USA, pp. 1–9 (2011)
3. Just, M.A., Carpenter, P.A.: A theory of reading: from eye fixations to comprehension. Psychol. Rev. **87**(4), 329–354 (1980)
4. Rayner, K.: Eye movements in reading and information processing: 20 years of research. Psychol. Bull. **124**(3), 372–422 (1998)
5. Duchowski, A.T.: Eye Tracking Methodology: Theory and Practice. Springer-Verlag New York Inc., Secaucus (2007)
6. Binkley, D., Davis, M., Lawrie, D., Maletic, J.I., Morrell, C., Sharif, B.: The impact of identifier style on effort and comprehension. Empir. Softw. Eng. **18**(2), 219–276 (2012)
7. Sharif, B., Maletic, J.I.: An eye tracking study on camelCase and under_score Identifier styles. In: 2010 IEEE 18th International Conference on Program Comprehension, pp. 196–205 (2010)
8. Sharafi, Z., Soh, Z., Gueheneuc, Y.-G., Antoniol, G.: Women and men - different but equal: on the impact of identifier style on source code reading. In: 2012 IEEE 20th International Conference on Program Comprehension, ICPC, pp. 27–36 (2012)
9. Turner, R., Falcone, M., Sharif, B., Lazar, A.: An eye-tracking study assessing the comprehension of C++ and Python source code. In: Proceedings of the Symposium on Eye Tracking Research and Applications, New York, NY, USA, pp. 231–234 (2014)
10. Crosby, M.E., Scholtz, J., Wiedenbeck, S.: The roles beacons play in comprehension for novice and expert programmers. In: 14th Workshop of the Psychology of Programming Interest Group, pp. 58–73 (2002)
11. Fan, Q.: The effects of beacons, comments, and tasks on program comprehension process in software maintenance. Ph.D. Dissertation, University of Maryland at Baltimore County, Catonsville, MD, USA (2010)
12. Hansen, M.E., Goldstone, R.L., Lumsdaine, A.: What makes code hard to understand? CoRR, vol. abs/1304.5257 (2013)
13. Busjahn, T., Schulte, C., Sharif, B., Simon, Begel, A., Hansen, M., Bednarik, R., Orlov, P., Ihantola, P., Shchekotova, G., Antropova, M.: Eye tracking in computing education. In: Proceedings of the Tenth Annual Conference on International Computing Education Research, New York, NY, USA, pp. 3–10 (2014)
14. Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J., Schulte, C., Sharif, B., Tamm, S.: Eye movements in code reading: relaxing the linear order. In: International Conference on Program Comprehension, pp. 255–265 (2015)