An Eye-tracking Study Assessing the Comprehension of C++ and Python Source Code

Rachel Turner
Department of Computer Science and Information Systems
Youngstown State University
raturner@student.ysu.edu

Michael Falcone
Department of Computer Science
University of Southern California
mfalcone@usc.edu

Bonita Sharif, Alina Lazar
Department of Computer Science and Information Systems
Youngstown State University
{bsharif, alazar}@ysu.edu

Abstract

A study to assess the effect of programming language on student comprehension of source code is presented, comparing the languages of C++ and Python in two task categories: overview and find bug tasks. Eye gazes are tracked while thirty-eight students complete tasks and answer questions. Results indicate no significant difference in accuracy or time, however there is a significant difference reported on the rate at which students look at buggy lines of code. These results start to provide some direction as to the effect programming language might have in introductory programming classes.

CR Categories: D.3 [Programming Languages], F.3.3 Studies of Program Constructs

Keywords: eye-tracking study, reading source code, C++, Python

1 Introduction

Does programming language affect one's ability to learn and comprehend source code? Which language should instructors use to teach programming? Is there a specific language that is more effective and efficient for certain types of developers (novices) compared to others? These questions have gathered much debate among programmers and instructors in the software engineering community. In this paper, we present a study aiming to understand if and how programming language affects the way programmers are able to comprehend simple source code.

We focus on two prevalent languages: C++ and Python. We chose C++ because it is a classic language that has maintained its popularity over the years. Proponents of Python state that it is simpler and more elegant than C++ [Agarwal and Agarwal 2005]. This study empirically evaluates whether this is indeed the case based on programmers' visual effort and comprehension of code. According to Glass [Glass 2002], programming should be taught by first reading existing programs rather than writing them (traditional teaching method). There have been some studies done in the classroom to determine if a certain language is better by looking at student grades [Enbody et al. 2009]. These types of studies provide information after the task has completed and do not give any insight into what transpired while the subject was actually performing the task. In this paper, we try to determine, via a controlled experiment, if there is a difference in the way programmers read and comprehend source code given a particular programming language.

We use an eye-tracker to unobtrusively record programmers' eye movements as they are performing tasks in 1) analyzing and explaining source code (overview task) and 2) finding bugs in source code (find-bug task). Results on task accuracy, task speed, and visual effort are reported. We also report differences in gaze patterns between novices and non-novices. These results can provide insight into the ways in which programming language affects analyzing and debugging code, which can have a direct impact on both academia and industry. In academia, using a programming language that is known to be more comprehensible can help students better learn core programming concepts. In industry, comprehensibility can impact language choice for new projects. The research questions this paper addresses are:

- RQ1: Does programming language affect the effectiveness and efficiency of solving overview and bug finding tasks?
- RQ2: Is there a difference in visual effort while reading and analyzing source code in C++ vs. Python?
- RQ3: Is there a difference in eye gaze behavior between novices and non-novices across C++ and Python?

2 Experimental Design

The goal definition template by Wohlin [Wohlin et al. 1999] is used to describe the experiment. The experiment seeks to *analyze* two programming languages (C++ and Python) *for the purpose of* evaluating their impact in overview and bug finding tasks *with respect to* effectiveness (accuracy), efficiency (time), and visual effort *from the point of view of* the researcher *in the context of* students at Youngstown State University. The main factor being analyzed is the programming language used. A between-subjects design was used, where each subject was tested on either C++ or Python programs but not both. We chose this design to shorten total study time and increase the data points in each language. There are three dependent variables: accuracy, time, and visual effort. The data collection was done via a video camera and eye-tracking equipment. Based on the research questions presented above, four detailed null hypotheses based on each of the three dependent variables are given below.

- H_a: There is no significant difference in *accuracy* between C++ and Python programs for overview and bug finding tasks.
- H_t: There is no significant difference in *time* between C++ and Python programs for overview and bug finding tasks.

- H_{ve}: There is no significant difference in *visual effort* between C++ and Python programs for overview and bug finding tasks.
- H_e: There is no significant difference between novices and non-novices in terms of accuracy, time, or visual effort, for C++ and Python programs for overview and bug finding tasks.

Previous work has shown that visual attention triggers mental processes and that visual effort is directly linked to cognitive effort [Duchowski 2003]. It has also been shown that processing of visual information occurs during *fixations*—prolonged gazes on a fixed locations, whereas no such processing occurs during *saccades*—time between fixations [Duchowski 2003]. We used a Mirametrix S2 eye tracker for this study, which records fixations as well as screen-capture audio/video. It is a video-based binocular remote eye tracker with a 60Hz capture rate and 0.5 to 1.0 degrees of average accuracy.

2.1 Programs, Tasks, and Stimuli

We chose simple code snippets as stimuli that contained fundamental programming concepts such as if statements, un-nested loops, arrays, and basic input and output to screen. We wanted the subjects to complete the study in a short time frame of less than 15 minutes to avoid any issues with fatigue. A total of ten stimuli (five in C++ and five in Python) were used. Each stimulus falls into one of two task categories: overview or find bugs. See Table 1 for an overview of the tasks used. A replication package is available at *http://www.csis.ysu.edu/~bsharif/etra14*. Each Python stimulus was shorter than its C++ counterpart due to the inherent nature of the language. Figure 1 shows an example of one code snippet shown to subjects, in both languages. Font size was kept the same for each task across C++ and Python and was chosen such that the eye tracker could distinguish between each line of code. Tasks were presented to subjects as follows.

Overview task: "Describe accurately and completely what the code does in your own words. You may also state the output if possible"

Find Bug task: "State the line number(s) that the logical error is located on. Describe in words what the error is and how you would go about fixing it"

2.2 Dependent Variables

Accuracy refers to whether subjects answered the task correctly and was scored from 1 (completely wrong) to 5 (fully correct). A systematic rubric for assessing the answers was defined for both find-bug and overview tasks and is provided with the replication package. Time refers to the number of milliseconds spent answering each task. Visual Effort is measured by fixation duration and number of fixations on each area of interest (AOI). The first two measures for visual effort are defined for both overview and find-bug tasks. The latter two are for find-bug tasks.

- Fixation Count (TotalFC) – Total number of eye fixations on the entire stimulus
- Fixation Duration (TotalFD) – Total time (ms) of all fixations on the entire stimulus
- Fixation Rate on Buggy Line(s) (FRBugLine) – Total number of eye fixations on the lines of code.
- Fixation Duration on Buggy Line(s) (FDBugLine) – Total time (ms) of all fixations on the buggy lines of code.

A higher fixation count, duration, and fixation rate indicates more effort exerted by subjects to solve the task. For overview tasks, the subject reads the entire code to figure out what it does. For find-bug tasks, the subject reads a code description and then reads the code to determine where it does not meet the description. Even though both tasks involve reading, it is expected that the method of reading is different between the two tasks.

2.3 Participants

The subject pool consisted of a mix of 38 undergraduate and graduate students from Youngstown State University, recruited from beginning and advanced programming and problem solving courses. Participation was voluntary and no compensation was provided. Subject ages ranged from 18 to 45 years. Each subject was assigned to the C++ or Python group and classified as novice or non-novice based on their experience as indicated in answers to a background questionnaire administered one week prior to participation. There were 25 participants in the C++ group and 13 in the Python group, due to fewer programming and problem solving courses taught with Python as the primary language. All subjects had normal vision. Some wore contacts or corrective lenses but this did not affect the measurements of the eye tracker.

2.4 Study Procedure and Instrumentation

The study was conducted in accordance with IRB policies and procedures at a dedicated Usability Lab. On the day of the study, subjects were informed that the purpose of the study was to understand how programmers debug and comprehend source code. They were seated approximately 65 cm away from the screen. Before viewing a task, subjects were presented with a screen giving instructions on what the upcoming task was. The two overview tasks were presented first (in a random order) followed by the three find-bug tasks (in a random order). Subjects did not interact with the keyboard or mouse and all answers were collected verbally. A short post questionnaire collected data about question difficulty, clarity, time needed, and familiarity with the tasks.

Table 1: Overview of tasks and stimuli used in the study. The bugs were located on the following lines given in parentheses: B1(4 and 7), B2(5), and B3(3) for C++ and B1(2 and 5), B2(3), and B3(3) for Python

Task ID	Task Category	Program Name	Program Description	Font Size (pt)	LOC in C++	LOC in Python
O1	Overview	Rounding	Prints Round down 5 times Prints Round up 6 times	31	12	7
O2	Overview	Factorial	Find the factorial of the number input	27	13	12
B1	Find Bug	Age	Prints an appropriate statement based on entered age	32	10	8
B2	Find Bug	Count by 5	Count by 5 from the entered number 10 times	34	9	6
B3	Find Bug	Print array	Print all items in an array	32	5	4

Prints an appropriate statement based on their entered age.

```
1  age = input("Enter Age: ")
2  if ( int(age) >= 16):
3     print ("Legal to drive but not
4             drink")
5  elif ( int(age) >= 21 ):
6     print ("Legal to drink")
7  else:
8     print ("Underage Minor")
```

Prints an appropriate statement based on their entered age.

```
1  int age;
2  cout << "Enter Age: ";
3  cin >> age;
4  if ( age >= 16)
5    cout << "Legal to drive but not
6            drink";
7  else if ( age >= 21 )
8    cout << "Legal to drink";
9  else
10    cout << "Underage Minor";
```

Figure 1: Stimulus B1 for Python (left) and C++ (right) with the program specification shown on top.

3 Experimental Results

The collected data consisted of a single screen capture video and a sequence of raw eye fixations in XML format for each subject. Some records had to be discarded due to technical and data integrity issues. Video and fixations were recorded continuously over all stimuli, so we defined task boundaries and AOIs a posteri using a tool built in Visual Basic .NET. This tool allowed us to define task boundaries as well as measure fixation counts and total fixation duration within each AOI. The tool is freely available at *https://github.com/seresl/EyeAnalyzer*. Descriptive statistics are shown in Figure 2 across all tasks cumulatively. Data from two participants was lost due to technical difficulties and data from the first stimulus had to be thrown out for two subjects due to user error. A total of 38 participant data records were retained and used. To compare results between groups, we used a linear mixed-effects regression model (that works well on unbalanced data) as well as a Mann-Whitney non-parametric test.

3.1 Accuracy, Time and Visual Effort

The two rows in Table 2 present the results for accuracy and time using the non-parametric Mann Whitney test. The median accuracy of the C++ group is slightly higher compared to the Python group. With respect to total accuracy for all the five tasks, the results indicate no difference between C++ and Python. The same can be said about the total time for all five tasks. This indicates that we cannot reject null hypotheses H_a and H_t. We also ran the test on individual task categories. This also did not uncover any significant differences. The linear mixed-effects regression test did not yield significant or interesting results. The four rows in Table 3 present results for the fixation counts, fixation durations, and fixation rate and duration on buggy lines (for bug tasks). These are shown combined for all five tasks. There is no significant difference for fixation counts and fixation durations between C++ and Python. However, we did find a significant difference (p-value=0.036) between the fixation rate on buggy lines of code (when looking at all three bug related tasks) between C++ and Python.

A linear mixed-effects regression model was also created along with a further breakdown analysis on each of the three bug tasks. This regression model confirms significance for the fixation rate on buggy lines of code (p-value =0.043) and also shows that the first find-bug task (B1) resulted in a difference between C++ and Python groups. However, no significant difference was found for fixation durations on buggy lines of code. We can only reject null hypothesis H_{ve} with respect to the rate at which subjects fixate on buggy lines of code. With respect to B1, the fixation rate for Python (mean: 0.347) was higher than the fixation rate for C++ (mean: 0.286). This means that the C++ group had fewer eye fixations on the buggy line of code compared to the Python group. See Figure 1 for the two B1 task programs as shown to the subjects.

3.2 Secondary Factor Interactions

We also wanted to determine if there were any interactions between subject ability: novice and non-novice on the results. There were 17 novices and 8 non-novices in the C++ group. The Python group had 10 novices and 3 non-novices. We again employed a linear mixed-effects regression model, using two explanatory variables: programming language (C++, Python) and ability level (novice, non-novice). The test found statistically significant differences between novices and non-novices in accuracy of bug tasks (p-value=0.002, st. error=0.412, t= 2.032), with non-novices performing 1.4 times better than novices. There was also a significant difference for the B1 find-bug task between C++ and Python with respect to novices (p-value=0.038) for the visual effort measure *FRBugLine*. Novices tend to have a higher fixation rate for Python compared to C++ on buggy lines of code in the B1 task. Within the C++ group, there is a difference (p-value=0.04) between novices and non-novices with respect to the fixation duration on buggy lines of code for find-bug task B1. Also, the accuracy of find-bug tasks were higher for the Python group. Among novices, the gap between find-bug task accuracy for C++ and Python was almost non-existent. Based on the above results, we can reject the null hypothesis H_e which means that there is a significant difference between novices and non-novices with respect to accuracy and fixation rate, and fixation duration on buggy lines for find-bug tasks.

Table 2: Un-paired Mann-Whitney p-values (alpha=0.05) for overall (all tasks) accuracy and time

Dependent Variables	U	p-value (2-tailed)
Accuracy	186.500	0.466
Time	158.000	0.902

Table 3: Un-paired Mann-Whitney p-values (alpha=0.05) for overall (all tasks) fixation counts, fixation durations, fixation rate on buggy lines and fixation duration on buggy lines.

Dependent Variables	U	p-value (2-tailed)
TotalFC	172	0.782
TotalFD	169	0.854
FRBugLine (for find-bug tasks)	94	0.036 *
FDBugLine (for find-bug tasks)	124	0.242

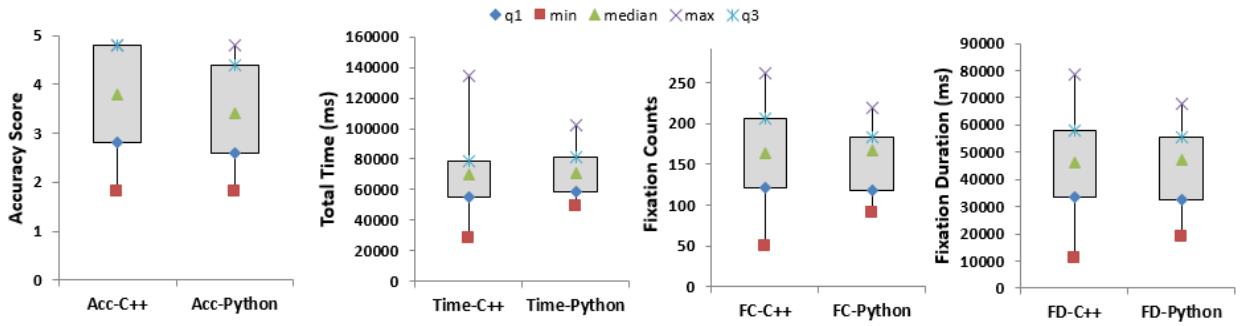

Figure 2: Accuracy, time, fixation counts and fixation duration box plots for C++ and Python

4 Discussion

Total accuracy for all tasks was higher for the C++ group. This was true for both novices and non-novices. With respect to time, the Python group took longer to solve all the tasks. Non-novices took longer in Python than novices. In the C++ group, novices took longer than non-novices. Novices tend to have higher total fixation counts in C++ compared to Python whereas non-novices had lower total fixation counts in C++. The total fixation duration for C++ and Python was almost same for non-novices whereas for novices, C++ had a slightly higher total fixation duration. For fixation rate on all the buggy lines of code, novices had a lower fixation rate compared to non-novices for Python. Among novices, Python had a higher fixation rate than C++. Finally, for fixation duration on all buggy lines of code, non-novices had mostly same fixation durations in both C++ and Python however, novices had higher fixation durations in Python.

It is possible that the non-novices were more accustomed to C++ and thus took longer in Python or this could also be attributed to the choice of the programs and the smaller Python sample. A larger study is called for to mitigate these threats. We now revisit the research questions posed in the Introduction. We did not find any significant differences in accuracy and time between C++ and Python (RQ1). We did find a difference in visual effort between C++ and Python but only with respect to how often subjects fixate on buggy lines of code (RQ2). There was also a significant difference in eye gaze behavior between novices and non-novices across C++ and Python (RQ3). However, given all of these results combined, we cannot yet recommend one language over the other for comprehending simple code such as that seen in this study. This study only represents a first step in understanding learning differences between languages.

5 Related Work

There are no eye tracking studies comparing C++ and Python source code. Very recently, Hansen et al. [Hansen et al. 2013] look at what parts of Python code are found most difficult. However, they do not compare Python and C++ and the results of the eye-tracking part of the Python study are yet to be published. Crosby and Stelovksy explored the way subjects viewed an algorithm and discovered that the eye movements of experts and novices differ in the way they looked at English and Pascal versions of an algorithm [Crosby and Stelovsky 1990]. Bednarik et al. investigated the visual attention of experts versus novices in debugging code [Bednarik and Tukiainen 2008]. No significant correlation was found between patterns of eye movement and performance in using the debugger. Uwano et al. also studied eye gaze patterns of five individuals while they were detecting defects in source code [Uwano et al. 2006]. This study was replicated by [Sharif et al. 2012]. Recently, [Binkley et al. 2013] study the impact of identifier style (i.e., camel case or underscore) on code reading and comprehension using an eye-tracker and found camel case to be an overall better choice.

6 Conclusions and Future Work

We present the results of an empirical study looking at accuracy, speed, and visual effort of subjects reading short C++ and Python code to complete overview and bug finding tasks. We found no statistical difference between C++ and Python with respect to accuracy and time, but did find significant difference between C++ and Python for fixation rate on buggy lines of code for find-bug tasks. As future work, we are conducting another study focusing on the main differences in the programming constructs of C++ and Python done using the within-subjects design, where each subject sees both C++ and Python code.

References

AGARWAL, K.K. AND AGARWAL, A. 2005. Python for Cs1, Cs2 and Beyond. *Journ. of Comp. Sciences in Colleges 20*, 262-270.

BEDNARIK, R. AND TUKIAINEN, M. 2008. Temporal Eye-Tracking Data: Evolution of Debugging Strategies with Multiple Representations. In *Symposium on Eye Tracking Research & Applications (ETRA)* ACM, Georgia, 99-102.

BINKLEY, D., DAVIS, M., LAWRIE, D., MALETIC, J.I., MORRELL, C. AND SHARIF, B. 2013. The Impact of Identifier Style on Effort and Comprehension. *ESE Journal 18*, 219-276.

CROSBY, M.E. AND STELOVSKY, J. 1990. How Do We Read Algorithms? A Case Study. *IEEE Computer 23*, 24-35.

DUCHOWSKI, A.T. 2003. *Eye Tracking Methodology: Theory and Practice*. Springer-Verlag, London.

ENBODY, R.J., PUNCH, W.F. AND MCCULLEN, M. 2009. Python Cs1 as Preparation for C++ Cs2. In *SIGCSE 2009* ACM, Chattanooga, Tennesee, USA, 116-120.

GLASS, R.L. 2002. *Facts and Fallacies of Software Engineering*. Addison-Wesley Professional.

HANSEN, M., GOLDSTONE, R. AND LUMSDAINE, A. 2013. What Makes Code Hard to Understand? *ArXiv e-prints*.

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 *ETRA*, Santa Barbara, CA, 381-384

UWANO, H., NAKAMURA, M., MONDEN, A. AND MATSUMOTO, K. 2006. Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. In *ETRA,* ACM Press, San Diego, 133-140.

WOHLIN, C., RUNESON, P., HÖST, M., OHLSSON, M.C., REGNELL, B. AND WESSLÉN, A. 1999. *Experimentation in Software Engineering - an Introduction*. Kluwer Academic Press.