Looking at Programmers*

Angela Edwards
Clemson University
Computer Science Department
Clemson, South Carolina, USA

Kiamber McCrorey
Clemson University
Computer Science Department
Clemson, South Carolina, USA

Khayla Williams
Clemson University
Computer Science Department
Clemson, South Carolina, USA

ABSTRACT

In this paper, we measure the effectiveness of code troubleshooting based on a student's classification. Troubleshooting techniques of freshmen will be compared to that of sophomores, and the techniques of juniors will be compared to the troubleshooting techniques of seniors. We tracked the eye movements of participants while they located and identified any errors presented in the different given coding samples.

CCS CONCEPTS

• **General and reference** → **Cross-computing tools and techniques**; *Surveys and overviews*; *General conference proceedings*; *Experimentation*; • **Hardware** → *Sensor devices and platforms*; Sensor applications and deployments;

KEYWORDS

eye tracking, expert, novice, error detection, LaTeX, text tagging

ACM Reference Format:
Angela Edwards, Kiamber McCrorey, and Khayla Williams. 2018. Looking at Programmers. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, Article 4, 5 pages. https://doi.org/10.475/123_4

1 INTRODUCTION

Computer Science (CPSC) is one of the fastest growing and sustained majors at Clemson University. The major and its subsets offer a wide range of opportunities for students including research and co-ops. Clemson's supercomputer, the Palmetto Cluster, is ranked as one of the top five supercomputers owned by a University in the country and the School of Computing is not far behind. Clemson takes a research first approach when teaching the art of code and introduces students to a wide range of languages throughout the curriculum.

In fall 2014, freshman Computer Science majors at Clemson had two options, start in CPSC 1010 and learn C or start in CPSC 1040 and learn Python. Under the direction of Chris Plaue, Clemson's computer science department added classes such as CPSC 1060, Intro to JAVA, and 1070, Programming Methodology. These classes have slightly altered the way that students in the major learn and retain troubleshooting information. For example, if your base language is Python, it is unlikely that you will look for a missing semicolon first as a possible source of an error. However, the participant started in C this is a much more plausible scenario.

Time in the major will always play an important part of in this study. A student with three to four years listed as a Computer Science major will be considered an expert for the purposes of this study. A person who has been in the major for anywhere between one month and twenty-three months will be considered as a novice for the purpose of this study. It is believed that exposure to different languages and possible errors over time will have an effect on this study, however, this study was designed so that anyone with basic knowledge of C, Python and JAVA syntax could complete all modules.

It is predicted that foundational principles of troubleshooting will be used by every participant regardless of status as expert or novice. We predict that participants will look for syntactical errors first and then began to address logical errors. We further predict that expert participants will look at the comments after they have determined whether or not syntactical errors are present. Lastly, we predict that the language order of the stimuli will have no effect on troubleshooting methods.

2 BACKGROUND

2.1 How Do Humans Edit

First, we will discuss how humans edit academic writings since there are much more studies on editing writing than there are on how to troubleshoot blocks of code. Academic writing "refers to a style of expression that researchers use to define the intellectual boundaries of their disciplines and their specific areas of expertise" [res]. Unlike academic writings, programming languages are understood by computers and are written in statements and blocks, rather than sentences and paragraphs.

Humans can edit academic writings by first editing for academic rigour, next by reducing redundancy in their papers, then editing for consistency and signposting and linking, and lastly, they can proofread. When editing for academic rigour you ensure your writing has evidence that supports your claims and that you wrote down exactly what you intended to. After that, ensure that your writing does not contain any unnecessary explanations, duplications, or irrelevant material. When editing for consistency, make certain that your use of tenses, voice, and style are similar throughout the paper. As well, when it comes to signposting and linking make sure you have let the reader know what he or she is expected to take from your paper. To make the editing process finalized, ensure you proofread your writing by checking for spelling, grammatical, and numbering errors [2].

Some studies have found that programmers read source code by skipping around the text looking for specific information. However,

*Produces the permission block, and copyright information

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 123-4567-24-567/08/06.
https://doi.org/10.475/123_4

Figure 1: Gazepoint Eye Tracker

one study showed that humans read source code in an identical to way to humans reading paper. This paper intends to focus more on eye movement patterns when finding errors in blocks of code.

2.2 Code Troubleshooting

Many studies discuss code troubleshooting, specifically Holmes et al. and Ko et al. found many of those studies to be focused on strategies and techniques used by programmers [5], [3]. "For example, some programmers follow an âĂIJopportunisticâĂİ strategy aimed at finding only the section of code that is needed during maintenance tasks [6]" [9]. The findings presented in this paper intent is to further explain this opportunistic strategy, especially since this form of comprehension differs between subjects.

3 METHODOLOGY

3.1 Subjects

For this experiment, we will use Clemson University students seeking a degree in Computer Science, Computer Engineering, Computer Information Systems or Mathematics. We expect that our participant pool will be made up of both male and female students. We will use a pre-participation questionnaire to categorize a participant as either a novice or an expert for the purposes of this experiment. We will have approximately fifteen participants for this study.

Because the nature of this experiment is a visual search task, participants must be able to demonstrate a base level of visual acuity and be able to discriminate various colors to complete the task. If participants were unable to meet this requirement, they were excluded from the experiment. Participants were recruited through emails and general announcements in various classes at Clemson University.

3.2 Apparatus

Figure 1. A Dell 22" monitor with a (1920 x 1080) resolution was used. The participants were seated at a distance of 24 inches from the monitor. The GazePoint eye-tracker was mounted under the display to pick up eye movement and pupil diameter. This can be seen in Figure 1. The sampling rate is 60Hz with a latency of 16ms and an accuracy of 0.5-1.0 degrees.

Example 1

```
//this function finds the size of an array
int main(){
   double arr[] = {11, 22, 33, 44, 55, 66};
   int n

   n = sizeof(arr) / sizeof(arr[0]);
   printf("Size of the array is: %d\n", n);
   return 0;
}
```

Figure 2: Syntactical error in C.

Example 2

```
//this function gets input from a user using scanf
int main(){
        int x;

        printf("Input an integer\n");
        scanf("%f", &x);

        printf("The integer is: %d\n");
        return 0;
}
```

Figure 3: Logical error in C.

3.3 Design

The type of experimental design used in the investigation was a within-subjects design. Participants were given a different sources of code and asked to locate and identify any errors present in an unlimited time frame. The GazePoint software tracker was used to identify key eye positions and measure the fixation time on each section of code.

Once the participant locates the error they will instruct the researchers and then be introduced to a blank image to restart the process all over again with a new block of code. This will then be repeated for the next two images. Each participant will look at the stimulus in random order for unbiased purposes and to reduce fatigue. The experiment was designed to include small, simple codes that most participants should be familiar with.

3.4 Stimulus

Figures 2-5

3.5 Procedure

Each participant will be greeted in the lab and read a script about the study. They will then read an informational letter approved by the Clemson University Institutional Review Board. The participant will be given the opportunity to ask any questions he/she may have

Example 3

```
// this program gives Fibonacci's number for a scanned in value
int main(){
        int val;
        scanf("%d" , val);

        int n = 0;
        for(i = val, i > 1, i-){
                n = n(i);

}
```

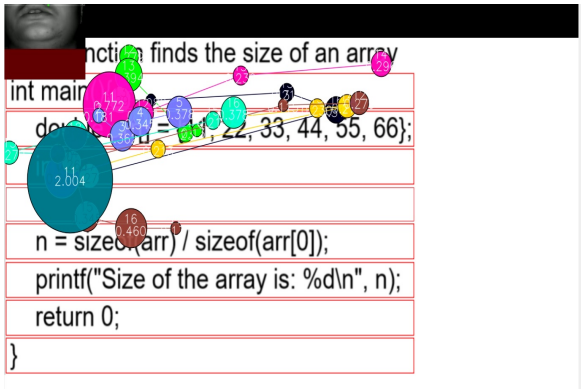
Figure 4: Complex code with multiple errors.

Example 4

```
//this function generates random numbers
int main() {
      int i, n;
      printf("Five random numbers between 1 and 100000\n");
      for (i == 1; i <= 5; i++) {
          n = rand()%100000 + 1;
          printf("%d/n", &n);
      }
      return 0;
}
```

Figure 5: Logical and Syntactical errors in C.

about the experiment. Once the participant has finished reading the letter, they will answer demographic questions about their age, gender, or any visual impairments that may skew the results of the experiment. Participants will then be set up on a 22 inch Dell computer and fitted to the Gazepoint eye tracker. To increase the accuracy of the data collected from the eye tracker, the user will be asked to calibrate the eye tracker using an image of nine numbered circles. After calibration, the participant will view exerts of code sampling and either verify that the would work or determine any errors that may be present in the code. The participant will then be instructed to click on the screen using the computer mouse to identify any errors. The stimuli for this study will be presented to each participant in a randomized order with a filler image between each code exert. The filler will be displayed for 10 seconds in order to give the participant both a visual and mental break. If the participant cannot identify anything wrong with the exert, they can give a verbal response "pass" and will be moved onto the next excerpt of code. Each participant will have 60 seconds to identify the error(s) in the excerpt. If the participant is confident that they have completed a module and identified all of the errors present, they will be instructed that they can hit the space bar to proceed. The participants will complete this process for all four (4) code samples.



1.jpg

Figure 6: Tracking patterns of eye movement

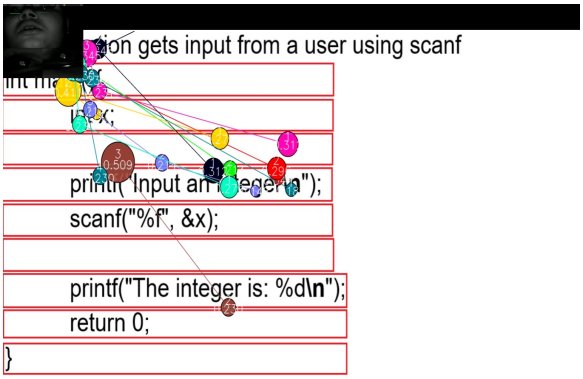
4 RESULTS

Eye tracking data was collected for 18 participants, but only 17 trials of data were analyzed for their gaze data. After a post study conversation, we determined that one participant was ineligible for the study due to falling outside of the participant criteria. Pre-study surveys were collected to obtain background information about participants in the study and this information helped to reduce as many variables as possible in student experiences.

The data that we collected was exported from Gazepoint Analysis for us to examine in depth for statistical significance of any kind. One thing we noticed while analyzing our data was that participants looked at our chart in a zigzag manner. Contrary to our hypothesis, most of our participants began the experiment by reading the comments, however another similarity among participants, regardless of experience level fell in line with our hypothesis. This similarity is based on the identification of syntactical errors. As shown in figures number 10 and 11, the time elapsed between the start of stimuli and identifying the first syntactical error was significantly lower than for that of identifying logical errors. We also noticed a high saturation of fixation points within loops and conditional statements when participants searched for logical errors. Additionally, we noticed that expert level participants spent significantly less time searching for errors. The average time that an expert spent on a module was 30 seconds whereas the average time spent for novices was equivalent to 41 seconds.

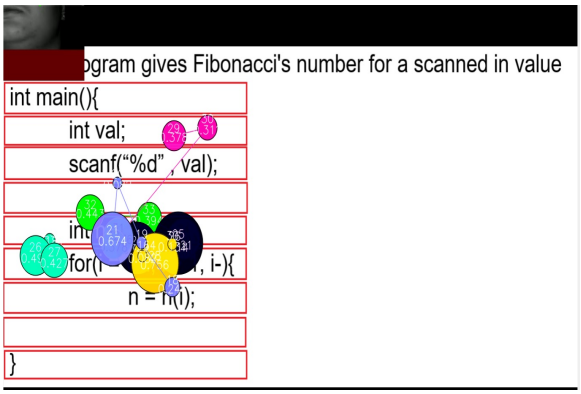
Although time was not originally a variable that we accounted for nor was of interest to us we did notice a significant trend in the time data. Both novices and experts were generally able to identify all errors present in the code within the 60-second margin, however, most experts spent significantly less time on each module. In contrast, both novices and experts were able to identify the first on every module on average in under thirty seconds. All participants identified syntactical errors first for every code exert.

Our results seem to support a narrative where the basic skills to troubleshooting are learned during the first few courses of Computer Science taught at Clemson University. Although older, expert level participants found the errors faster and with more confidence, novice level participants possessed the necessary skills to dissect and troubleshoot the errors presented in this study.



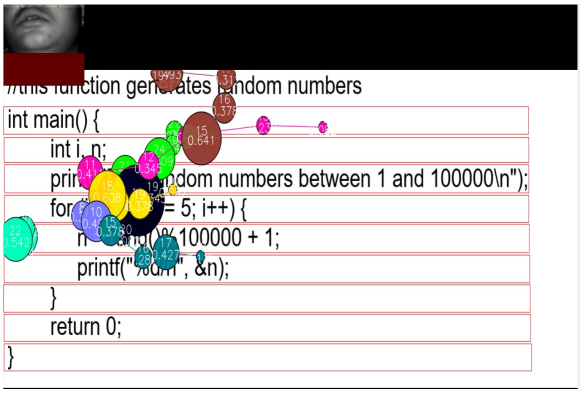
2.jpg

Figure 7: Within 2 seconds of a new module starting most participants gaze shifted to the comments



3.jpg

Figure 8: Concentrated fixation points within loop statements.



4.jpg

Figure 9: Concentrated fixation points on loop declarations.

Based on our observations, if we were to perform this experiment again, we would eliminate the timed component of this study and give participants unlimited time to search for errors. Although we do not believe that the time element had an adverse effect on

Elapsed Before First Error Identification.png

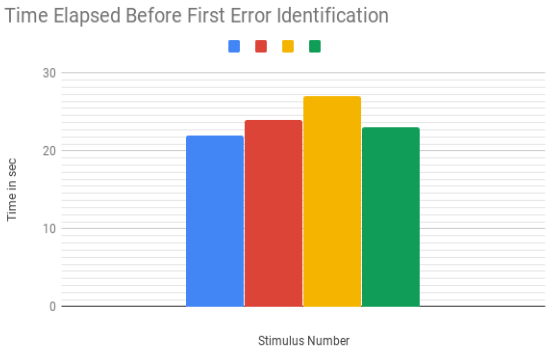


Figure 10: Average time elapsed before experts identified the first error in each exert

Elapsed Before First Error Identification (1).png

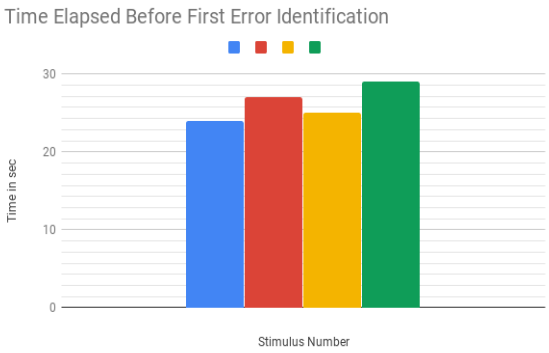


Figure 11: Average time elapsed before novices identified the first error in each exert

Time Spent er Stimulus.png

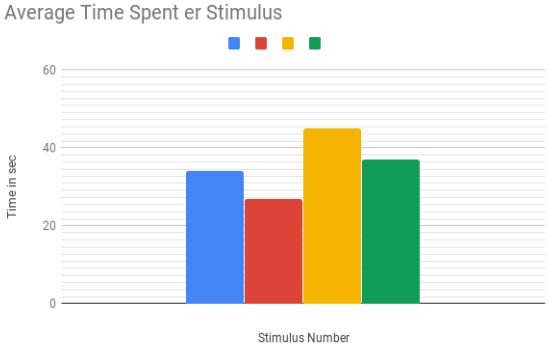


Figure 12: Average time experts spent on each module

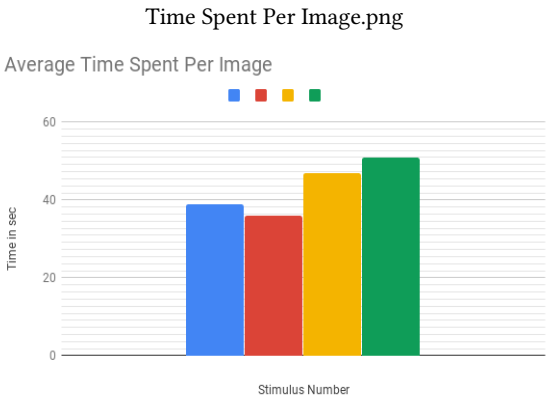


Figure 13: Average time novices spent on each module

our results, we note that it is a previously unaccounted for and unnecessary variable that could influence participant behavior.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Duchowski for teaching Computer Science 4120 and helping guide us through this experiment. The authors would also like to thank each of the volunteer participants for taking part in this study. Lastly, we would like to thank Clemson University's Internal Review Board for the diligence and commitment to excellence.

REFERENCES

[res] Research guides: Organizing your social sciences research paper: Academic writing style.

[2] (2013). The art of editing.

[3] A. J. Ko, A. Myers, M. J. C. and Aung, H. H. (2006). An exploratory study of how developers seek relate and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987.

[4] Crosby, M. E. and Stelovsky, J. (1990). How do we read algorithms? a case study. *Computer*, 23(1):25–35.

[5] Holmes, R. and Walker, R. (2013). Systematizing pragmatic software reuse. *ACM transactions on software engineering and methodology*, 21(4):20:1–20:44.

[6] J. Brandt, M. Dontcheva, M. W. and Klemmer, S. R. (2010). Example-centric programming: integrating web search into the development environment. In *CHI Conference : We are HCI : conference proceedings, Atlanta, Ga, USA, April 10-15, 2010*, pages 513–522. Association for Computing Machinery.

[7] Laderman, M. (2014). Clemson university moves up in global supercomputer rankings.

[8] Lakhotia, A. (1993). Understanding someone else's code: Analysis of experiences. *Journal of Systems and Software*, 23(3):269–275.

[9] Rodeghero, P. and McMillan, C. (2015). An empirical study on the patterns of eye movement during summarization tasks. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. IEEE.

[10] Sharif, B., Falcone, M., and Maletic, J. I. (2012). An eye-tracking study on the role of scan time in finding source code defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications*, ETRA '12, pages 381–384, New York, NY, USA. ACM.

[8] [9] [10] [4] [7] [res]