HOW NOVICES READ SOURCE CODE

by

Leela Krishna Yenigalla

Submitted in Partial Fulfillment of the Requirements

for the Degree of

Master of Computing and Information Systems

YOUNGSTOWN STATE UNIVERSITY

December, 2014

HOW NOVICES READ SOURCE CODE

Leela Krishna Yenigalla

I hereby release this thesis to the public. I understand that this thesis will be made available from the OhioLINK ETD Center and the Maag Library Circulation Desk for public access. I also authorize the University or other individuals to make copies of this thesis as needed for scholarly research.

Signature:

Leela Krishna Yenigalla, Student Date

Approvals:

Bonita Sharif, Thesis Advisor Date

John Sullins, Committee Member Date

Feng Yu, Committee Member Date

Sal Sanders, Associate Dean of Graduate Studies Date

Abstract

Every expert was once a novice. It takes a lot of dedication and time for novices to master any programming language. What if there is an improvement in design as well as method of teaching that can foster the process of learning. In the field of Computer Science education, design of course curriculum as well as tools for learning has always been important. It is a great challenge to design and integrate new methods to teach programming, so that any Computer Science novice can learn and master necessary skills and knowledge. In this thesis, we conduct an empirical study using eye tracking equipment to understand how novices read source code in the context of two programming classes. Our main goal is to begin to understand the strategies and techniques they use to read source and their 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 results also show that we did not see any shift in the stage of learning the novices are currently at, indicating that there might be more than one course that needs to be taken over the course of a few years to realize the shift. We call for more studies to further learn about this shift.

Acknowledgements

First of all I would like to express my true appreciation to my advisor, my mentor, Dr. Bonita Sharif. I appreciate her vast knowledge and skill in many areas and her assistance in writing my thesis report. I could not have imagined having a better advisor and mentor for my Masters. I will forever be thankful.

I thank my committee members Dr. John R Sullins and Dr. Feng George Yu for their support and guidelines.

I thank my parents and my brother for their love and encouragement. I thank all the members who directly or indirectly helped me to complete my research work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Programming is an intertwined process of reading and writing (Busjahn and Schulte 2013). 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 programming comprehension is rarely considered (Busjahn, Schulte, and Busjahn 2011) . In general, reading plays a crucial role in tasks such as debugging, analysis, maintenance, comprehension, and most importantly learning. Novices have a tough time in learning programming languages. If we understand how novices read source code, and what hardships they face during initial learning, we can design better tools, and working environments. We are doing this research to explore the potential that lies in analysis of reading process. Eye tracking along with think aloud protocol is considered as a viable tool of research in code reading studies. In this research, we are using a carefully designed eye tracking study, accompanied by pre and post test questionnaires for comprehension to obtain information on the linking process between interpretation and comprehension. This can be very useful in improving our knowledge on design of an educational framework.

1.1 Motivation

Previously research (Busjahn and Schulte 2013), was done to find out the importance of code reading and comprehension in teaching programming. Researchers

interviewed instructors based on their experience to find, the importance of code reading in five categories such as conceptualization, occurrences, and effects of successful code reading, challenges for learners, as well as approaches to facilitate code reading. The results of these interviews pretty much indicate code reading is connected to comprehending programs and algorithms, or algorithmic ideas, as well as details, like e.g. semantics of constructs. But at the same time there is not much knowledge about the reading and comprehension process of learners. If we knew more about this process, suggestions for learning tasks could probably be improved. However, so far, we do not know much about good reading strategies. Hence we are doing this research to find out, the techniques novices use to solve programming comprehension. Analyzing the data of novices helps to design better strategies, because a possible means to foster learning is to teach reading directly, including reading strategies.

1.2 Contributions

The main contribution of this thesis is an empirical study that assesses how students, especially novices read source code. Data collection was done using two methods: online questionnaires and an eye tracker (hardware and software). Subjects were students at Youngstown State University. This project was conducted, to find strategies novices use while doing comprehension. The ultimate goal of this thesis is to develop better teaching strategies specifically targeted to novices and how they learn. However, in order to do this, we first need to conduct several studies to determine individual behavior and determine if any differences exist before we can generalize this process.

1.3 Research Questions

The following are our research questions that we seek to answer.

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

1.4 Organization

This thesis is organized as follows. The next chapter gives a brief introduction to eye tracking and related work. Chapter 3 presents the details of the experimental setup for the study. Chapter 4 discusses observations and results. Chapter 5 concludes the thesis and presents future work.

CHAPTER 2

BACKGROUND AND RELATED WORK

This chapter talks about an overview of program comprehension and empirical studies conducted using eye tracking equipment. Eye tracking has always been a useful methodology in many fields, where human computer interaction takes place. This chapter discusses about use of eye tracking in various fields.

2.1 An Eye tracking Overview

An eye tracker is able to detect where a person is looking at on the screen. Visual attention refers to the focus on a particular location on the screen. It is known that visual attention triggers mental processes in order to comprehend and solve a given task (Just and Carpenter 1980). Effort put visually is directly linked to the cognitive effort (Just and Carpenter 1980). *Fixations* and *saccades* are two main types of eye gaze data. A fixation is when the eye stabilizes on a 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 (Rayner 1998; Duchowski 2007). We now present a subset of studies done in some field starting with software engineering.

2.2 Eye-tracking Studies in Software Engineering and Program Comprehension

Software engineering (SE) is about developing, maintaining, and managing complex and high quality software systems, in a cost economical and efficient way. In the field of software engineering, development of knowledge about the use of technologies can be accumulated by systematic studies of empirical research methods. (Sjoberg, Dyba, and Jorgensen 2007). It is essential to know, how useful different software engineering technologies, are for different actors performing different tasks, on different working environments, such knowledge about SE will help to develop, new technology and can play a key role in decision making in SE industry. Development and modification of existing technologies such as process models, methods, techniques, tools or languages, and evaluation of such technologies, in interaction environments, like organizations, teams, projects. Surveys are exclusively suitable for answering questions about what , how much, and how many , as well as questions about how and why (Sjoberg, Dyba, and Jorgensen 2007). Now we discuss some studies that are relevant to this research.

Sharif et al. study the impact of identifier style (i.e., camel case or underscore) on code reading and comprehension using an eye-tracker (Binkley et al. 2012; Sharif and Maletic 2010b). They find camel case to be an overall better choice for comprehension. Sharafi et al. (Sharafi et al. 2012) conduct an eye tracking study to determine if gender impacts the effort, time, and ability to recall identifiers. Guehéneuc (Guéhéneuc 2006) investigated the comprehension of UML class diagrams. Jeanmart et al. (Jeanmart et al. 2009) conducted a study on the effect of the *Visitor* design pattern on comprehension using an eye tracker. Sharif et al. (Sharif and Maletic 2010a) also conducted an eye

tracking study assessing the role layouts have in the comprehension of design pattern roles. Yusuf et al. (Yusuf, Kagdi, and Maletic 2007) used eye-tracking equipment to assess how well a subject comprehends UML diagrams. Busjahn et al. (Busjahn et al. 2014) talk about the relevance of eye tracking in computer education.

Recently, Turner and Sharif et al. (R. Turner et al. 2014) conducted a comparison study between C++ and Python to assess effect of programming language on student comprehension of source code. The motivation behind this study is to find whether programming language affects one's ability to learn and comprehend source code and which is the ideal language to use for teaching programming. Hence the authors chose C++ and Python, because C++ is a classic language and remained popular over the years, while many also believe Python is simple and more elegant than C++. Many researchers believe that programming should be taught by first reading existing programs rather than writing them. Eye tracker is used as an evaluation tool for this research study, eye gazes are tracked while 38 subjects complete tasks like analyzing and explaining code, finding bugs in source code. Results on task accuracy, task speed, and visual effort are reported. These results will provide insight into various ways in which programming language affects analyzing and debugging code. A total of 10 stimuli, five in C++ and five in Python were used. Each stimuli falls into one of two task categories: overview or find bugs. To explain in detail, for overview task subjects need to describe accurately and completely what the code does, in their own words, as well as the output as needed. For the find bug task, they had to state the line number(s) where the logical error is located. They also needed to describe in words what the error is and how they would fix it.

Measures of visual effort for overview are fixation count and fixation duration and for find bug tasks fixation rate on buggy lines, and fixation duration on buggy lines. Parameters like higher fixation count, duration, and fixation rate indicates more effort given by subjects to solve the task. The main difference between two tasks is for overview task, subjects reads the entire code to figure out what it does. For find bug tasks, the subject reads to find bugs in the code to determine where it does not meet the description, even though both the tasks require reading, the method of reading is different. Dependent variables for this study are accuracy, time, total fixation count, total fixation duration, fixation rate of buggy lines, and fixation duration of buggy lines. As a result, researchers found no statistical difference between C++ and Python with respect to accuracy and time, there is a significant difference between C++ and Python for fixation rate on buggy lines of code for find bug tasks. To contribute more for this research, as a future work, authors of this paper are going to do another study focusing on the main differences in constructs of C++ and Python. This time each subject sees both C++ and Python code.

Eye tracking helps to understand difficulties, strategies, and helps develop new tools in the field of computer science. It is also used during usability studies of new websites to determine the user friendliness of the websites, to check where the user is looking, to find out user expectations. In education, it can be used to develop techniques to teach students based on the data gathered during empirical studies. It is also useful to test IDE's like NetBeans, Eclipse, and Visual Studio, to determine the easiness in creating JAR files, classes, integrating new codes etc.

Eye tracking is also used for software traceability (Sharif and Kagdi 2011) . Based on the research conducted by computer science experts eye tracking can be used to find the effects of text-based, graphical, or UML program representations, syntax and language features, programming paradigms, the behaviors and strategies of a learner's reading, understanding, writing and debugging tasks. Challenges for beginners, like what makes code hard to understand, what obstacles impair their understanding and use of programming concepts. Evaluation of tools for static, dynamic program visualization, to instantly provide feedback of programming environment based on eye movements. In the field of computer science empirical studies play a key role in understanding the usability. The detailed data collected using eye tracking, helps to create advance learning tools like IDE's for making programming easy and accurate. Eye tracking is a well-established instrument in usability testing with a large corpus of analyses, metrics.

Visual cueing could be employed in an IDE, if students look too long at the wrong section of code, or thrash their gaze over the entire program without focusing on any particular part (Busjahn et al. 2014). Use of eye tracking in computer science can lead to new perspectives of teaching, especially teaching code reading. Eye tracking can give a glimpse on how individuals conceptualize, perceive the given computational references. Gaze analysis offers intriguing prospects for future research in computer science. Without the need of subjective report, cognitive process of the person can be recorded using the eye tracker. Hence this research tool provides a new quality and directness, along with a much finer data granularity, to observe cognitive processing.

In 2010, Fan presented her dissertation on the effect of beacons, comments, and tasks on program comprehension (Fan 2010). This provides a lot of food for thought and suggest possible extensions of our work as well.

Human subjects are involved in studies related to validation software visualizations, Conventional measures such as accuracy and performance time are the parameters of assessment. Eye tracking data adds a new dimension to the assesment by providing an uynique opportunity to include measures of how exactly users use a software or tool. (Kagdi, Yusuf, and Maletic 2007). The data collected using eye tracking is useful for number of scenarios. Till now the main application has been usability tests of software , websites, UML files, assessment of novices and experts in the real time work environment.(Atterer, Wnuk, and Schmidt 2006)

Utilizing eye tracking as a part of the evaluation of web inquiry interfaces can give rich data on users information-seeking conduct, especially in the matter of user collaboration with diverse educational parts on a query items screen. One of the fundamental issues influencing the utilization of eye tracking in examination is the quality of caught eye movements. User evaluation permits specialists to get a wealthier understanding also to portray bits of knowledge into the distinctions in the middle of great and poor web seek interfaces. (Al Maqbali et al. 2013)

2.3 Eye tracking Studies in Physics

A part of physics education research deals with the differences between novices and expert's, specifically how they solve problems (Rosengrant 2010). Even though there have been a great deal of research take place to see the difference novices and experts

show in physics , but none of those studies focuses on where they exactly look while solving problems. Problem solving is not always finding numerical answer , it can include qualitative solutions ,such as using simulations to explain how a microwave works (Rosengrant 2010). As we know one of the main goals of education researchers is to narrow the difference between novices and experts (Rosengrant 2010). To make it possible, in earlier days, researchers used to do one-on-one interviews with students, where they used to give students a task and ask them to think aloud, so that they can determine their thought process while solving problems. Combination of think aloud protocol with eye tracking gives a new research method called gaze scribing, this method was created by David Rosengrant. Subjects will be asked to write their solutions on a graphic monitor. During the process they wear a head mounted eye tracker, both their audio and video will be recorded and further used for analysis. The initial study focuses on electrical circuits as there is a unique combination of qualitative understanding of representations as well as reasoning abilities. Researchers found the differences in problem solving techniques of experts and novices. Eye-tracking has been widely used for research purposes in fields such as linguistics and marketing.

However, there are many possibilities of how eye-trackers could be used in other disciplines like physics. A part of physics education research deals with the differences between novices and experts, specifically how each group solves problems. Though there has been a great deal of research about these differences there has been no research that focuses on noticing exactly where experts and novices look while solving the problems. Thus, to complement the past research, David Rosengrant created a new technique called

gaze scribing. Subjects wear a head mounted eye-tracker while solving electrical circuit problems on a graphics monitor. Researchers monitor both scan patterns of the subjects and combine that with videotapes of their work while solving the problems. This new technique has yielded new information and elaborated on previous studies (Rosengrant, 2010). These differences include, novices typically write down the known and unknown variables. Next, they use a backward inference technique a search for equations involving variables they think they can use. This technique is generally called as plug and chug.

Experts use a forward inference technique which mainly focuses on determining the concepts and key features of the problem, to find a solution to solve the problem. While categorizing novices do it by surface features of the problem, but experts still go with underlying concepts. While problem solving the way novices and experts get unstuck, when they get stuck during the process is also largely different. When novices get stuck they typically either manipulate equations or ask for help from outside to get them unstuck, whereas experts will check their solutions, possibly using methods such as multiple representations or alternate mathematical equations. One of the major conceptual challenge to novices is current in DC circuits, they also believe despite of the arrangement of resistors, parallel branches split the current equally through branches, and current get consumed when flowing through a circuit. Instead of relying solely on written and verbal responses, It is better to rely upon gaze scribing as it provides a good opportunity to analyze behavioral data. Differences in the scan paths, which are collected during the gaze scribing, provide new interpretations and much more clear understanding of differences between novices and experts.

Researchers found consistent patterns of wrong answers for conceptual physics questions. scan path differences between students who answer physics problems correctly and incorrectly can be investigated using an eye tracker. Fixation location, duration and order is compared between two groups. AOI's of the students of both groups are analyzed to teach students about the strategies of students who answered them correctly. (Madsen et al. 2012)

2.4 Eye tracking Studies in Aviation

Technical innovations in aviation and improvements in air traffic management are the biggest challenges aviation faces in 21 century (Hasse, Grasshoff, and Bruder 2012). Due to advancement, operators need to work on highly automated systems, thus requiring operators monitoring appropriately (OMA). Eye tracking is the main basis for determining the OMA, Operational monitoring includes using one's senses to observe and understand data acquired from different sources, like navigation instructions. Gaze movement data of the operators classify high and low performing subjects. According to models of adequate monitoring behavior difference between experts and novices can be stated based on the target-oriented attention allocation both in general and during monitoring phases such as orientation phase, anticipation phase, detection phase, and recheck phase (Hasse, Grasshoff, and Bruder 2012). Large number of psycho physiological and imaging studies indicate that the eye tracking data is an appropriate method for measuring the efficient and accurate visual information. Fixation counts can be used as a measure of the expectations and assumptions of the person fixation durations reflect information processing duration, and the total gaze duration per AOI is a measure

of the difficulty of recording the information viewed. Another important aspect where eye tracking is useful in aviation is aircraft inspection, Aircraft inspection is a very important task which assures safety and security of the air transport system. The two main types of aircraft inspection are visual inspection, non destructive inspection. Studies say that ninety percent of the aviation maintenance inspection is visual (Sadasivan et al. 2005) . Training aircraft inspection personnel is an essential and effective strategy, to obtain accurate inspection results. Important element of training is provision of feedforward information. Feedforward training is nothing but providing a prior report of information, which consists of special strategies, important precautions, lethal defects, particular location of defects. Offline training by experts is an effective way to train novices. Since experts adopt a better inspection strategies compared to novices, Providing analyzed eye tracking data of expert's to novices, will help them understand the cognitive processes, and adopt experts strategy. With the help of eye tracking equipment, we can record the point of interest of an expert inspector while performing an inspection task, in a virtual reality simulator. Analyzing the fixations of the expert's leads to visualization of their scan paths which allows us to display the inspectors' visual search, hence their cognitive strategy. A definitive research on eye tracking shows that sequences or scan paths changes with change in subjects' strategy when viewing a scene. This sequential model of visual attention is based on the sequential eye movements, and this visual attention sequence is explained in three stages, first, during the fixation information is processed, second, the visual attention is shifted to peripheral scene region (an area outside of the current fixation),third, an eye movement is programmed and executed to

the newly selected location. Finally, applications of eye tracking or scan path based feed forward training are potentially numerous, as it may be used for broader range of human activities involving skilled performance (Sadasivan et al. 2005).

Inspection and maintenance are crucial for safety and reliability of air transport system. To improve inspection performance, training is used as a primary arbitration strategy. Tools play a crucial role for inspector trainees, to improve their skills and to make training successful. A indicative eye tracking virtual reality system is developed by researchers, for recording eye, as well as head movements, and inspection time and realization time during the persons engagement in virtual inspection simulator. This virtual reality simulator consists of a head mounted displayed eye tracker, which records the progressive point of interest inside the simulator. (Duchowski et al. 2001)

In the year 2000 the Air Force Research Laboratory (in collaboration with BBN Technologies) launched the Agent-based Modeling and Behavior Representation Project (AMBR). The three parameters that subject to assessment using the eye tracker are performance, reaction time and subjective workload ratings. Tasks designed for assessment are little modified version of air traffic control task, in this every subject acts as a air traffic controller, responsible for supervising aircraft as they pass through a radar screen. For optimal assessment, all the subjects need to complete the tasks of a air traffic controller under time pressure. This leads to prioritize necessary actions and management of multiple objectives despite of frequent interruptions. Comparisons of this study are based on tasks, text display, color display. There is no alternative for the point of view that eye tracking can provide into these processes (Bartels and Marshall 2006).

Task dependent automation and adaptive interfaces help air traffic controllers to adapt to variating work loads. Main challenging factor in the application of these kind of intelligent working environment concerns with a question, what exactly operators doing in order to support and minimize automation surprises. Eye metrics play a key role in determining the tasks operator do in cockpit. Six eye metrics are assessed and used as a dependent variables, fixation saccade , scan path length , fixation duration, convex hull, fixation clusters, spatial density, nearest neighbor(ratio fixation distribution and random distribution) . (Imants and de Greef 2014)

2.5 Eye tracking Studies in Medicine

A challenging goal today is the use of computer networking and advanced monitoring technologies to extend human intellectual capabilities in medical decision making (Faro et al. 2010). Improvement of precision in eye movements capture, made eye tracker as a tool for vision analysis. Human eye tracker interface becomes more and more important in medical field as it allows doctors to increase diagnosis capacity. Client /server eye tracking system is a new method that provides an interactive way to monitor patients' eye movements based on the clinical test administered by physicians. Eye tracking in recent years have been used in human and computer interface research in usability evaluation, but also in both e-learning applications. (Faro et al. 2010). Characteristics that are useful to determine the health status and human vision like pupil diameter, right left eye position and their distance are calculated using the commercial eye tracker. Commercial and new generation eye trackers are easy to install, portable to move and robust in giving results. Advantages of eye tracker are, it allows offline

analysis of research results, processes gaze information for several purposes like discovering pathologies related to eye vision, retrieval of relationships with new pathologies, Training and support for new ophthalmologists. (Faro et al. 2010).

In medical field eye tracker works as oculomotor, which focuses on saccadic eye movements. Those movements can be used as a powerful experimental test of the brain functions and health conditions. Correlation between working memory, attention and eye movements is pointed out. Knowledge on how saccadic movements work, may support the study of neural system and highlight various neurological disorders. Patients suffering from diseases like Parkinson, diabetes, Alzheimer's, Parkinson's, Huntington diseases will have abnormalities in saccades. Tests that are widely used are perimetry, visual attention, eye velocity, and contrast sensitivity. Disorders defected by one of these tests, can be associated to a specific disease. Basic proposed system architecture consists of a hardware instrument, management software, human operators. A Tobii T60 eye tracker device at the patient side with a connection to a personal computer and another PC on the doctor side. Management software consists of an eye tracker which allows the remote analysis of gaze information. Finally the main features of the proposed system are low interaction at patient side because test handling is all performed by the client side at the doctor side, Configurable and powerful interaction at the doctor side, the eye tracker interface is very user friendly to provide accurate tests and create new vision tests, it also needs calibration and analysis like any other eye tracking test. (Faro et al. 2010)

2.6 Eye tracking Studies in Gaming

Eye tracking helps gamers to find out the understandability of games (J. Turner et al. 2014), gaming is a field where both mental physical aspects of the users should be considered. Hence any game that is about to release in the market needs to be assessed based on the eye tracking study. Normally there are two things that designers consider while conducting eye tracking test in gaming, who are the target audience and does it have capability to update constantly like every other game in the market. Eye tracking is the most efficient method to use to determine whether game is user-friendly and manageable and enjoyable. It tracks users eye movements and their mental aspects while playing the game. It also checks whether the instructions are useful and understandable. Eye tracking can be used as research tool to inform game design (J. Turner et al. 2014). Discreet gaze patterns of players eye movement varies with gender , skill, age, which results in future design of games. Based on the interest and ease of use of individual games can be similar games can be designed, to attract users. One of the main aspects to measure user experience of a game is Playability (Bernhaupt, Eckschlager, and Tscheligi 2007).

2.7 Eye tracking Studies in Usability Testing

In academic practice as well as in commercial sector, eye tracking research is used progressively to augment usability tests. Research suggests that, there is a solid correlation between usability issues and eye tracking patterns. Usability issues are not connected just to a single eye tracking pattern but to a definite series of patterns. This series of patterns seems to arise from various coping strategies that users develop when a

problem is experienced. With increase in usage of internet in to daily life, it is expected that the user experience of any website or software product should be a positive one. In a competition driven market, demand for usability analyses using eye tracking is developing rapidly (Ehmke and Wilson 2007). The visualizations of eye tracking data are recorded and analyzed by usability researchers, to pinpoint the perplexity from the user side, but also to identify the areas users are looking at. Expert usability researchers look at the following eye tracking data: long fixations which represent interest or confusion, and back track saccade - not looking at elements of a page, Scanning behavior rather than reading behavior, that is fixations and saccades not in left to right order with sweeps, back and forth between objects, first place the user looks, last place the user looks, when making a choice fixations back to one item then final scan before making choice, readings headings or subheadings (Ehmke and Wilson 2007).

2.8 Eye tracking Studies in Human Computer Interaction

Previously, eye based human computer interfaces mainly concentrated on making use of the eyes in traditional desktop settings. Recent improvement of interest in smart glass devices and low cost eye trackers, however gaze based techniques for mobile computing is becoming increasingly important (Pfeiffer, Stellmach, and Sugano 2014). A new paradigm called pervasive eye tracking introduced into human computer, human - human and wearable computer systems. This is continuous monitoring and eye based interaction 24/7. The main advantage of this application is the capability to track and analyze eye movements, anywhere, anytime. This leads to a new research for better understanding of visual behavior and eye based interaction. An interdisciplinary approach

of research is necessary to explore this field. Development of video based eye tracking technologies has lead to a new research in daily environments and wearable devices. Wearable cameras and devices such as Google glass are also gaining importance in recent years along side eye tracker components which have a huge importance in near future.

Due to the development of unobtrusive eye tracking technology, the potential scope for research in interactive environments has increased. Eye trackers have also become a lot cheaper and one can acquire a decent one for less than $150. Multi-user eye tracking is a reliable tool to study social factors in visual thinking and support collaborative interaction. Methods in human computer interaction and eye tracking are automated eye movement analysis, or evaluation of eye movement classification algorithms. Finally the research areas that evolved from eye tracking and human computer interaction are pervasive eye-based interaction, mobile attentive user interfaces, eye-based activity and context recognition, security and privacy for pervasive eye-tracking systems, eye tracking for specialized application areas, and eye-based human-robot and human-agent interaction.

CHAPTER 3

The Empirical Study

This chapter presents the details of the empirical study conducted as part of this thesis. It gives details on the experiment design, hypotheses, data collection, tasks, and participiants and how the study was instrumented.

3.1 Experiment Design

An overview of the experiment is given 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 before each phase was conducted. The novice was instructed to go through the code snippets shown to them. They were also asked about the level of difficulty and their confidence level of answering the questions related to the code snippets.

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)

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.

3.2 Hypotheses

Based on the research questions presented above four detailed null hypotheses based 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 count* between Phase 1 and Phase 2.

H_{fd}: There is no significant difference in *fixation duration* 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.

3.3 Participants

We recruited students from two classes in Fall 2014 at YSU. The first class was an object oriented 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 thesis. 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 the pre questionnaire. Figure 1 shows demographics of participants.

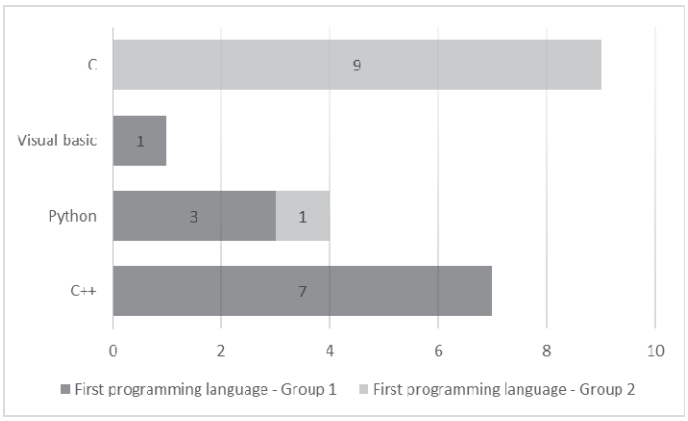


Figure 1. Descriptive statistics on first programming language in both groups

We can see that most of the students in Group 2 had C as their first programming language whereas it was C++ for Group 1. Also, Group 2's expertise in Java was slightly higher than Group 1's. Both groups in majority spent about an hour a month programming.

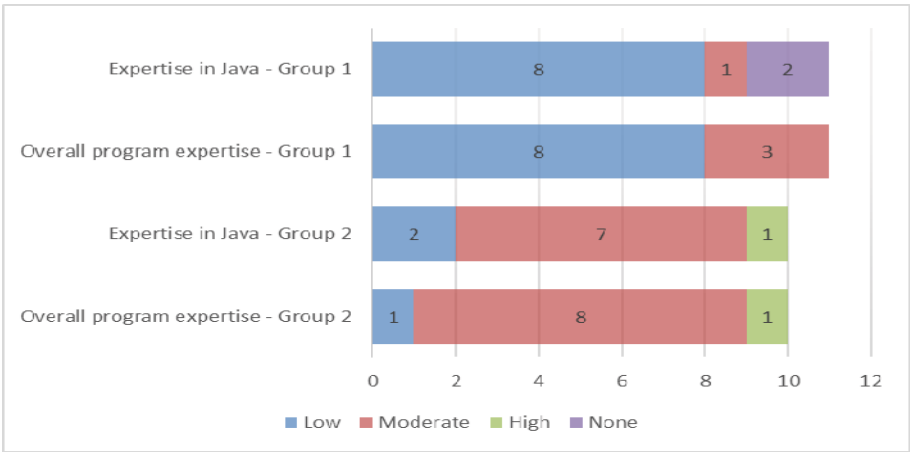


Figure 2. Descriptive statistics on Java expertise in both groups

Group 1 consists of 8 students of age between 18-22 years, and remaining 3 students are between 23-27 years. There are 2 female subjects and 9 male subjects in this

group. The average years of programming experience in Java is less than a year, whereas two students have no experience and one student has an experience between 1-2 years. In other programming languages, 7 students have 1-2 years of experience, 2 students have 3-5 years of experience, one student has less than 5 years of experience, another student has less than one year of experience in other programming languages. When the students are asked to rate their overall expertise in programming, 8 out of 11 students rated themselves as novices(low programming skills), 3 students rated themselves with medium level expertise. Particularly in Java, 8 students have basic knowledge (low), 2 of them are new to Java programming (no expertise), 1 student has moderate expertise (medium). 5 out of 11 students work on programming languages other than Java for less than 1 hour a month, 3 students work less than 1 hour a day on other programming languages, 2 students work for less than 1 hour per week, 1 student work more than an hour every day. When it comes to programming in Java, 4 out of 11 students work 1 hour per month, 3 students work less than an hour a day, 4 students work on Java for less than an hour per week.

In Group 2, apart from 1 student , all the subjects are between 23-27 years. 9 out of 10 students are male, Half of the students (5 out of 10) have less than a year of experience in Java, whereas 3 students have 1-2 years of programming experience in Java, one student is new to Java. When it comes to programming experience in other languages 5 out of 10 students have an experience between 1-2 years, 3 students have experience of 3-5 years, one student has an experience of less than a year, and the remaining subject is a novice. When the students are asked to rate their overall

programming skills, 8 out of 10 students rated themselves as moderate programmers (medium level of expertise), one student rated himself as an expert and another student rated himself as a novice. When it comes to Java 7 students rated themselves as moderate programmers, 2 as novices, one as an expert. Three out of 10 students work less than a hour per week on programming languages other than Java. 2 students work less than a hour per month, 3 students work less than a hour per day. 2 students work more than a hour per day on other programming languages. When it comes to working in Java, 4 out of 10 students work less than 1 hour per month. 3 students work less than 1 hour per week. 2 students work on java less than 1 hour a day. 1 student works more than 1 hour per day on Java.

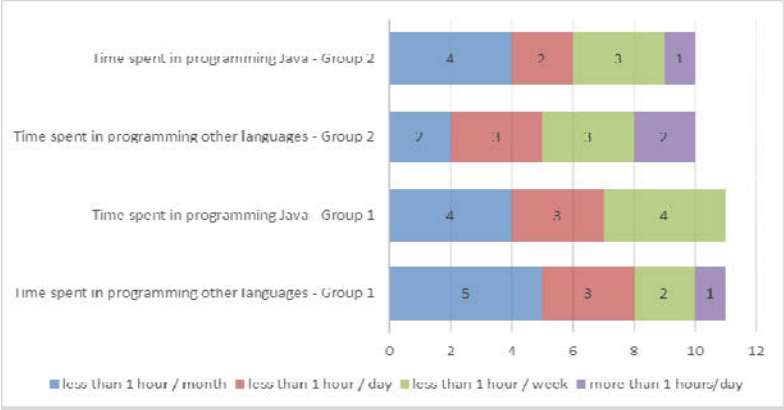


Figure 3. Time spent programming in Java and other languages in both groups.

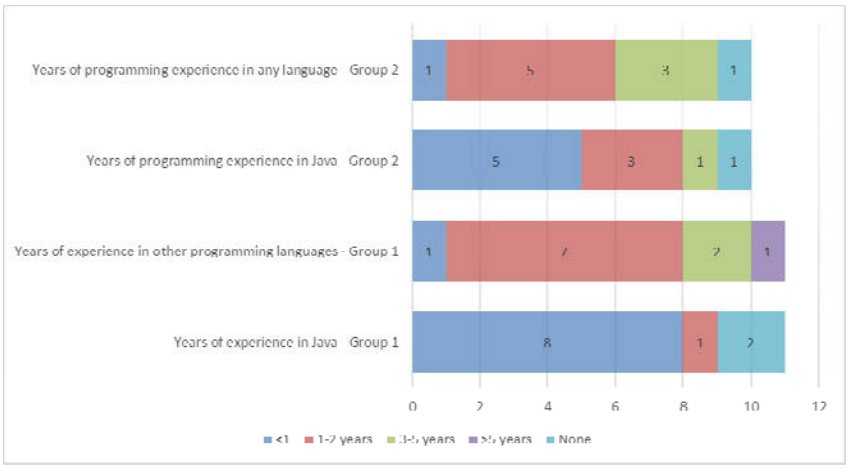


Figure 4. Programming experience in both groups

3.4 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 array of strings, static keyword, random, substring, static class methods. In Phase 2, the tasks invoved 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 including all background questions and post questionnaires can be found in the Appendix.

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 given to see if it was easier after a couple of months. So for

example, Primes program was compared with CheckString from phase 2 in the difficult category, StringProcessingDemo was compared with TextClass and Count was compared with PrintPattern.

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

Program Name	Task	Overview of the Program	Difficulty Level	Phase 1	Phase 2	LOC	Concepts Used
StringCheck	Output	string comparison	Easy	X		12	if/else statement, Strings
Primes	**Summary**	**finds all primes <=n**	**Difficult**	**X**		**31**	**nested for loop, if/else statement, nested if**
TestPassArray	Output	swapping array elements	Medium	X		25	parameter passing, call by reference, 2 methods
StringProcessingDemo	**Summary**	**change part of a string**	**Easy**	**X**		**15**	**substring, indexof, Strings**
Rectangle	Output	area of a rectangle	Difficult	X		24	parameter passing, constructor, this variable, 4 methods
Count	**Summary**	**number of times a letter occur in a string**	**Medium**	**X**		**21**	**for statement, if statement, returns an array, isLetter, charAt, toLowercase, 1 method**
TextClass	**Summary**	**change part of a string**	**Easy**		**X**	**9**	**string, indexof, substring, length, replace**
TestingCircle	Summary	exception handling	Medium		X	53	Exception handling, OO concepts, 2 classes, creating objects, throws vs. throw, parameter passing
CheckString	**Summary**	**check if string input is a palindrome**	**Difficult**		**X**	**23**	**while loop, if/else statement, parameter passing, Strings, charAt, length, 1 method**
DoSomething	Summary	selection sort	Easy		X	18	nested for loops, if statement, arrays, 1 method
PrintPattern	**Summary**	**prints three rows of stars in triangle**	**Medium**		**X**	**13**	**nested for loop, parameter passing, 1 method**
KeyboardPanel	Summary	draws a letter and moves it using arrow keys	Difficult		X	29	Gui and events, addkeylistener, keyevent, keyadapter, keypressed…

```
public class StringProcessingDemo
{
    public static void main(String[] args)
    {
        String sentence = "I hate text processing!";
        int position = sentence.indexOf("hate");
        String ending = sentence.substring(position + "hate".length( ));

        System.out.println("012345678901234567890123");
        System.out.println(sentence);
        System.out.println("The word \"hate\" starts at index " + position);

        sentence = sentence.substring(0, position) + "adore" + ending;
        System.out.println("The changed string is:");
        System.out.println(sentence);
    }
}
```

```
public class TextClass {
    public static void main ( String [ ] args ) {
        String text = "Hello World!" ;

        int positionW = text.indexOf( "W") ;
        int textLength = text.length () ;

        String word = text.substring ( positionW , textLength - 1 );

        System.out.print ( text.replace ( word , "Hello" ) ) ;
    }
}
```

Figure 5. Two comparable programs from Phase 1 (top) and Phase 2 (bottom)

3.5 Data collection

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.

In addition to the online questionnaires, we collected eye tracking data and audio/video recordings of subjects that did the study at Youngstown State University because we have access to an eye tracker at this location. We did obtain IRB approval and training before we began this study.

3.6 Eye-Tracking Apparatus

The Tobii X60 eye tracker (www.tobii.com) was used in this study at one location primarily at YSU. It is a 60Hz 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 inch monitor with screen resolution set at 1920 * 1080. The study was configured to use a dual monitor extended desktop setting. The first monitor was used by the experimenter to setup and initiate the study. The eye tracker records eye-gaze data and audio/video recordings of the entire study session on the second monitor. The eye gaze data includes timestamps, gaze positions, fixations and their durations, pupil sizes, and validity codes. In this study, only fixations and their durations are used and we setup the entired experiment in Tobii Studio.

3.7 Conducting the Study

The study participants were first required to fill out the background questionnaire. The test was conducted in the Software Engineering Research and Empirical Studies Lab (SERESL).The test can only be attempted by one student at a time as the lab can only have one student in at a time. The experiment goes like this: subjects arrive into the lab, they are provided with consent forms as well as an instruction sheet. Once they go through them, they will decide whether to participate in the test or not. They can withdraw themselves if they have any issue. Calibration will be done before starting the test, to make sure the subject's eyes are in sync with the eye tracker. Once the calibration

starts, subjects will see a red circle with a black dot in the middle. They need to focus on the black dot. Once the eyes of the subject is calibrated (see Figure 6 for an example of how good calibration looks like – If the green vector is too far away from the circle then the error is higher.), researcher can go ahead and start the session. First task will be a sample question just to get them familiarized with the process. Answers to the sample task are provided. Subjects are encouraged to ask questions, just to make sure that they understand the instructions. Solutions or results of the test are not given to subjects. The experimenter is always present with the subject to make sure the eyes are always being tracked which is determined via the Tobii eye status monitor. Once the recording starts, subjects cannot pause or end the session abruptly without completing it, unless it is an emergency and they want to opt out. In these cases, if they wished to continue we started a new session.

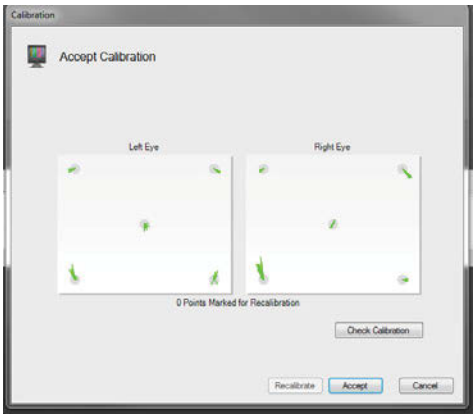


Figure 6. Good calibration.

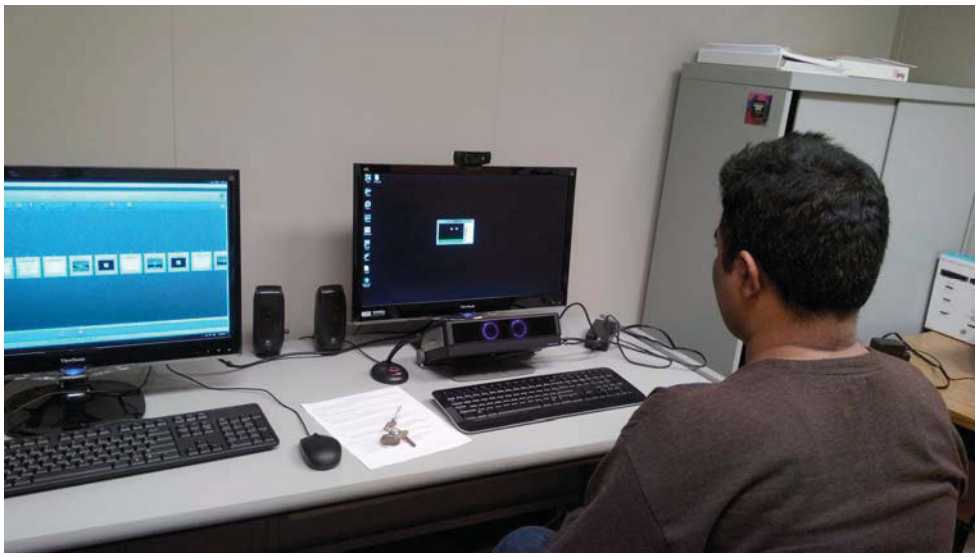


Figure 7. Work space of a person participating in the study. The screen on the left is for the experimentor, the right screen is used by the subject. The eye tracker Tobii X60 is seen at the base of the right screen.

CHAPTER 4

Results and Analyses

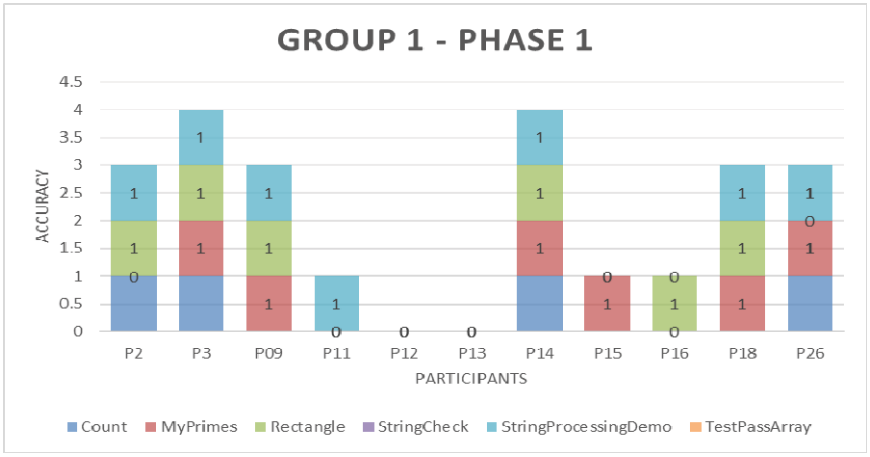
This chapter presents the results from our controlled experiment. Since our data is not normal and we have a small sample size, we use non-parameteric measures to determnine significance using the Wilcoxon paired test. Alpha is set at 0.05 that determines significance with a 5% error and 95% confidence.

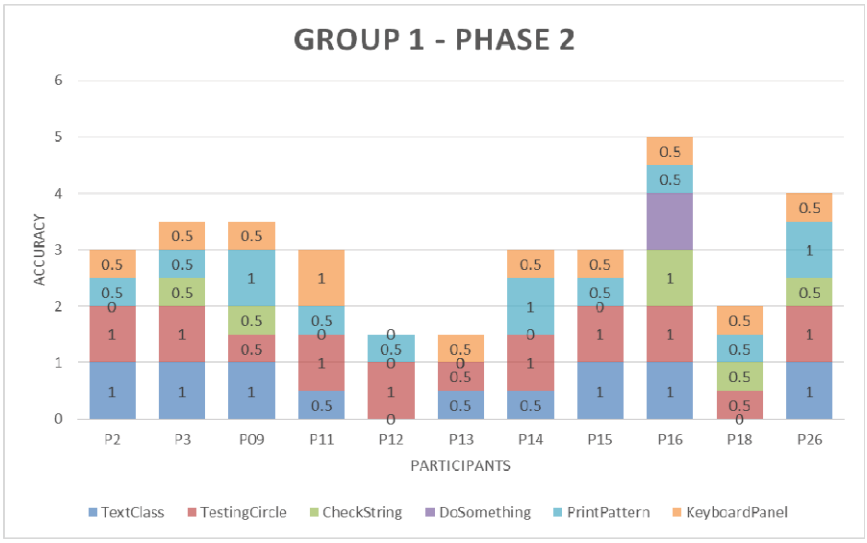
4.1 Accuracy

Each of the twelve programs were scored by the experimenter 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 (p-value=0.045). In Group 2, Wilcoxon test also shows that Phase 1 total accuracy was significant less than Phase 2 (p-value=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 Figure 8 for the descriptive statistics.





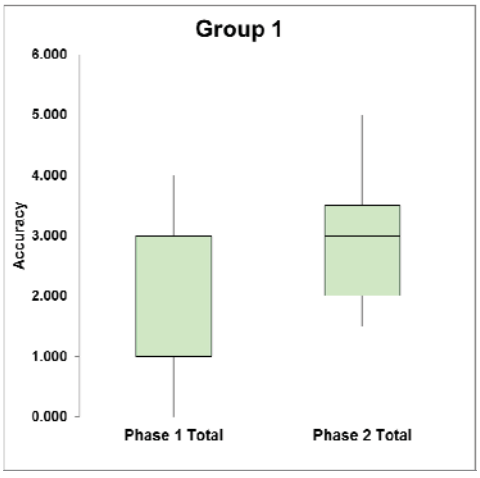
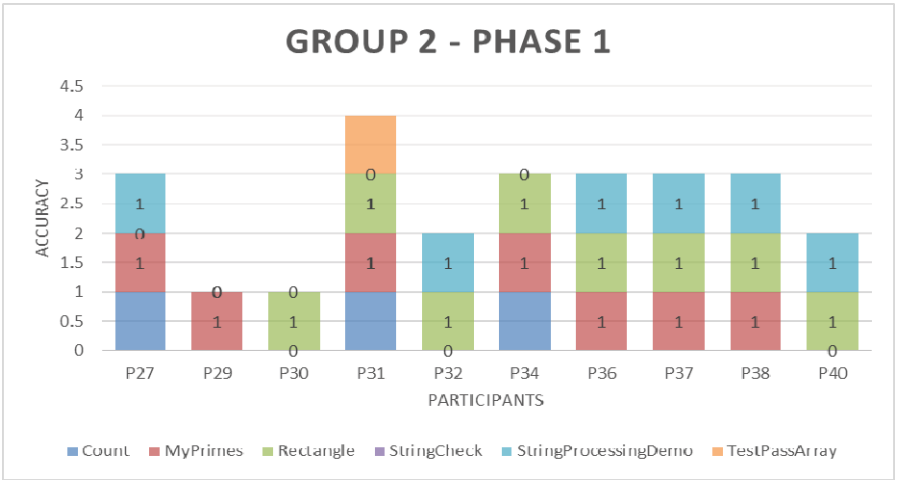
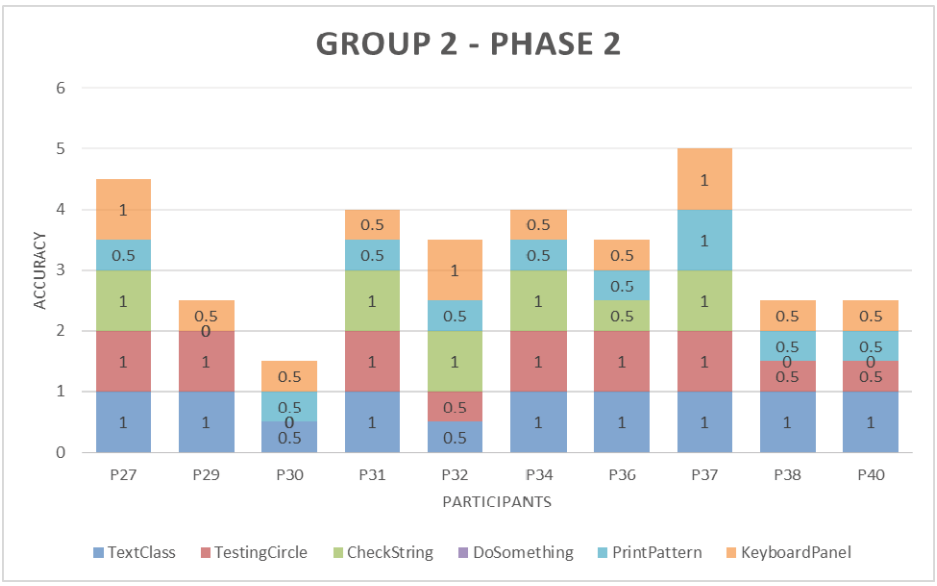


Figure 8. Results for Accuracy for Group 1 for both phases.





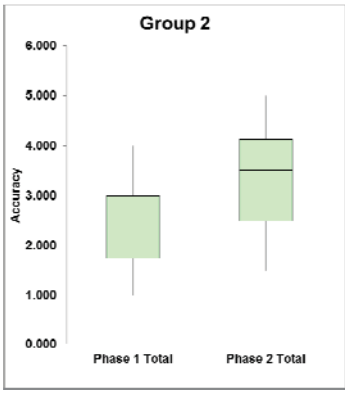


Figure 9. Results for Accuracy for Group 2 for both phases.

We notice the same trend in Group 2 as well. They scored higher in Phase 2. They also did slightly better when compared to Group 1 however, this difference was not significant.

4.2 Time

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).

- Count and PrintPattern (p-value=0.012) – Medium difficulty
- StringProcessingDemo and TextClass (p-value=0.051) – Easy difficulty

where the one in phase two took longer time to solve.

Each question was timed via Tobii Studio. The descriptive statistics for time are given below from Figure 10 through Figure 13 grouped by each program. We notice how the StringCheck program that was the easiest was done in the least amount of time with not much variation between the subjects. The MyPrimes 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 subjects 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.

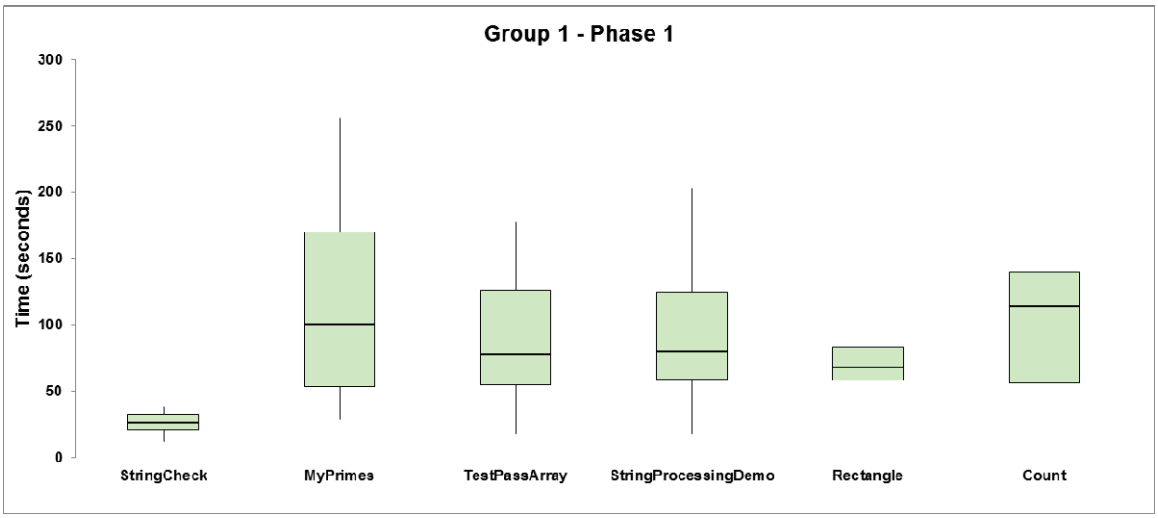


Figure 10. Results for Time for Group 1 – Phase 1

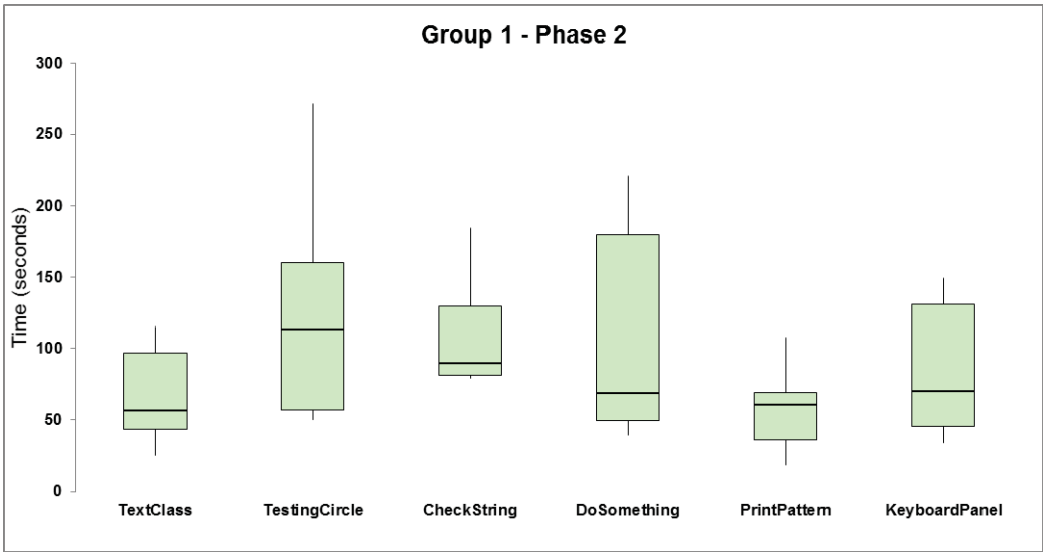


Figure 11. Results for Time for Group 1 – Phase 2

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.

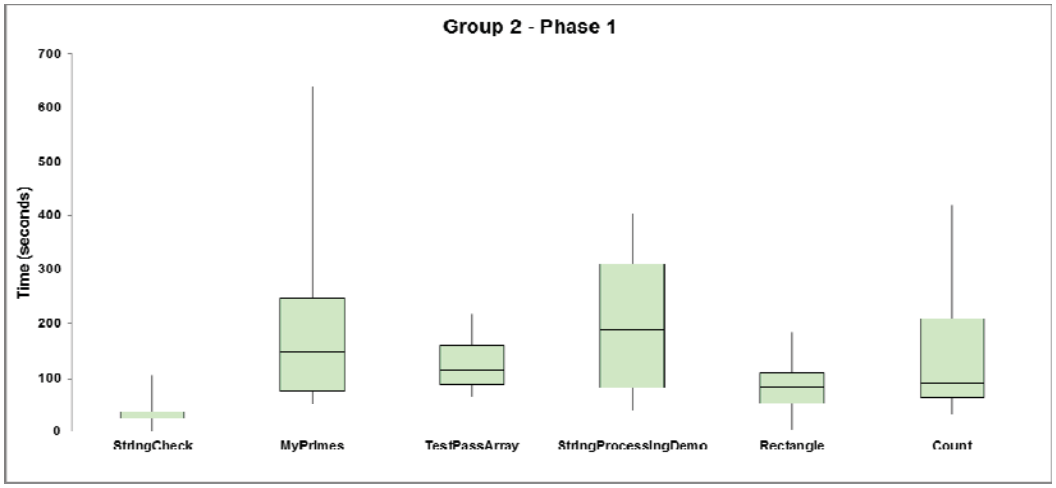


Figure 12. Results for Time for Group 2 – Phase 1

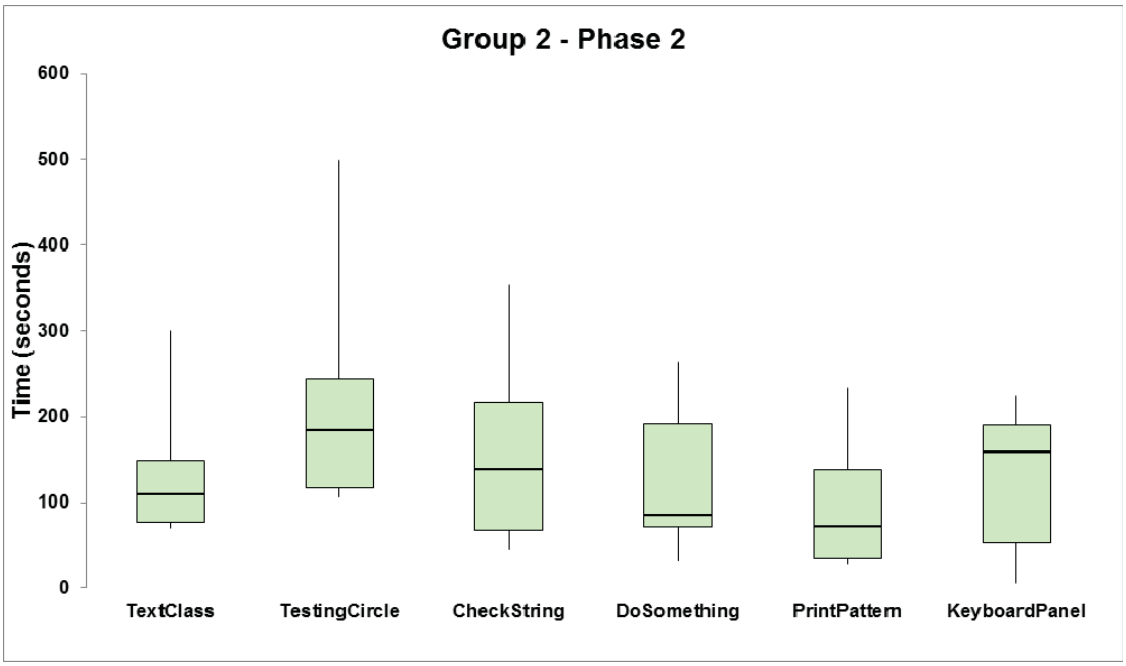


Figure 13. Results for Time for Group 2 – Phase 2

4.3 Creating Areas of Interest

Before we calculate fixation counts and durations we need to define areas of interest for which to collect them in. Figure 14 below shows fixations of a particular user on the Count program. The next figure (Figure 15) shows the AOIs for one of the programs used in the study. The AOIs comprise of 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 on blank space.

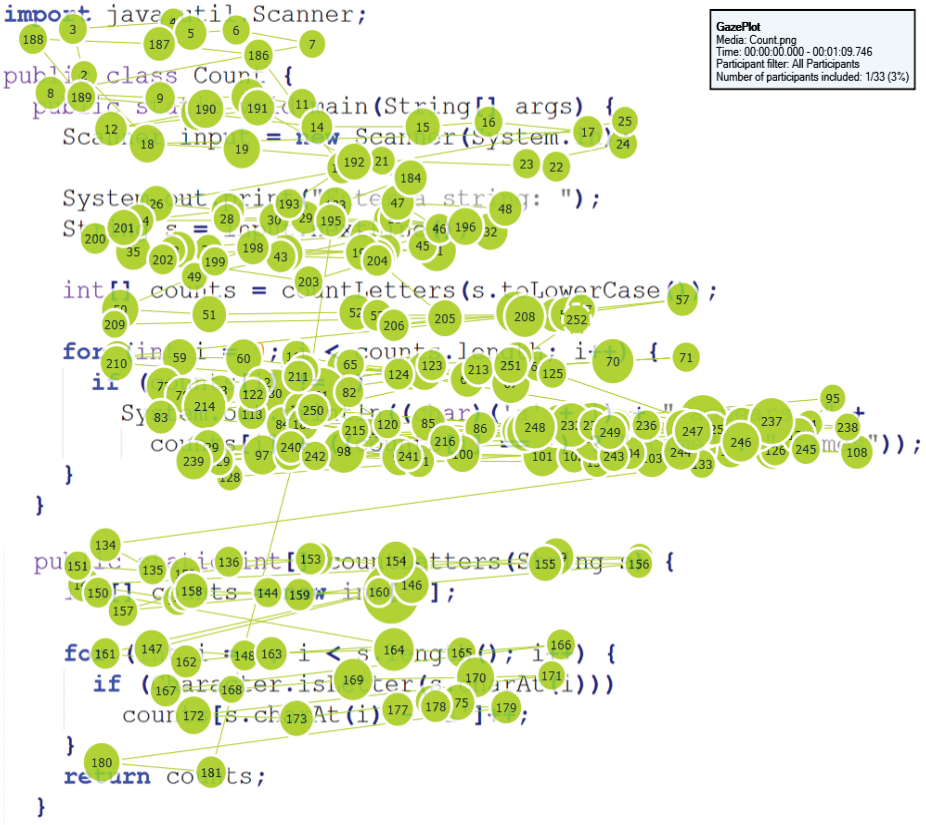


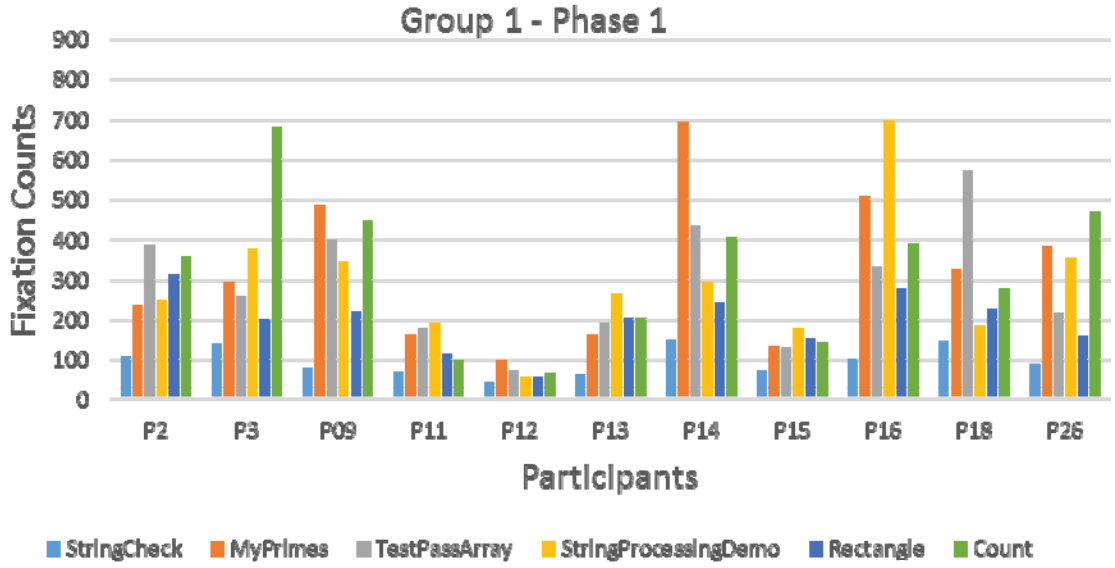
Figure 14. Fixation counts and durations for the Count program. The circle indicates a fixation and the radius of the circle indicates the fixation duration. Almost line level granularity is achieved. A scanpath is a collection of ordered fixations in sequence.

Figure 15. Areas of Interest

4.4 Fixation Counts

The fixation count denotes the number of fixations spent in total within the AOIs for each program. For example, if a program has 10 lines, we would total the fixation counts for each of those 10 lines. This resulting number would give us the fixation count for that particular program.

The Wilcoxon test did not report any significant results for fixation counts or fixation durations. Figure 16 and Figure 17 show the fixation counts for each of the twelve programs in Phase 1 and Phase 2 for Group 1 and Group 2 respectively. In Phase 1, the highest fixation counts were on MyPrimes, 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 (P16). None of the students got the DoSomething program correct in Group 2.



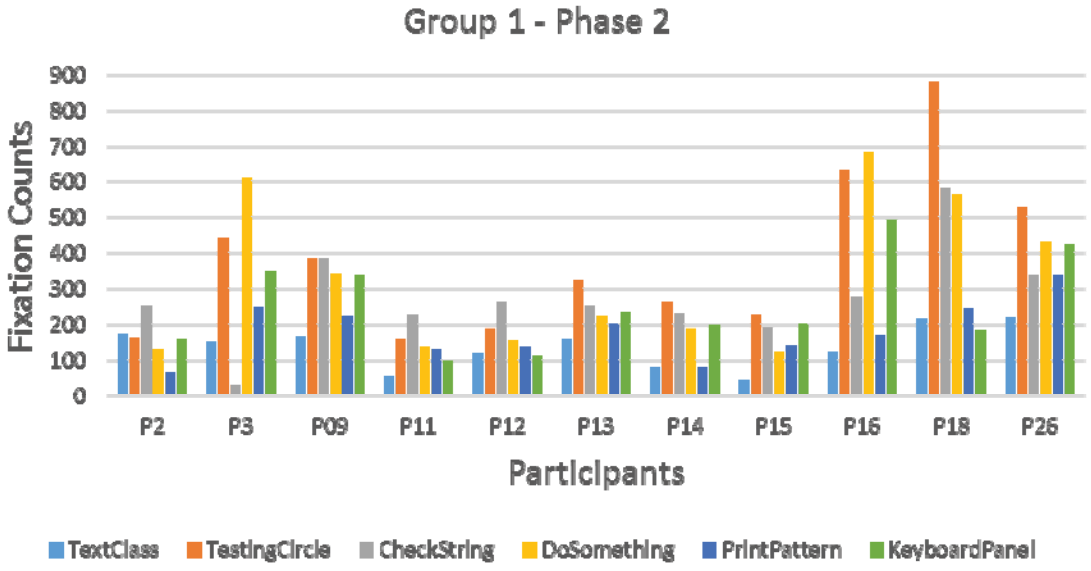


Figure 16. Fixation Counts for Group 1 both Phases

The same trend is observed in the fixation counts in Group 2 for Phase 1 and Phase 2.

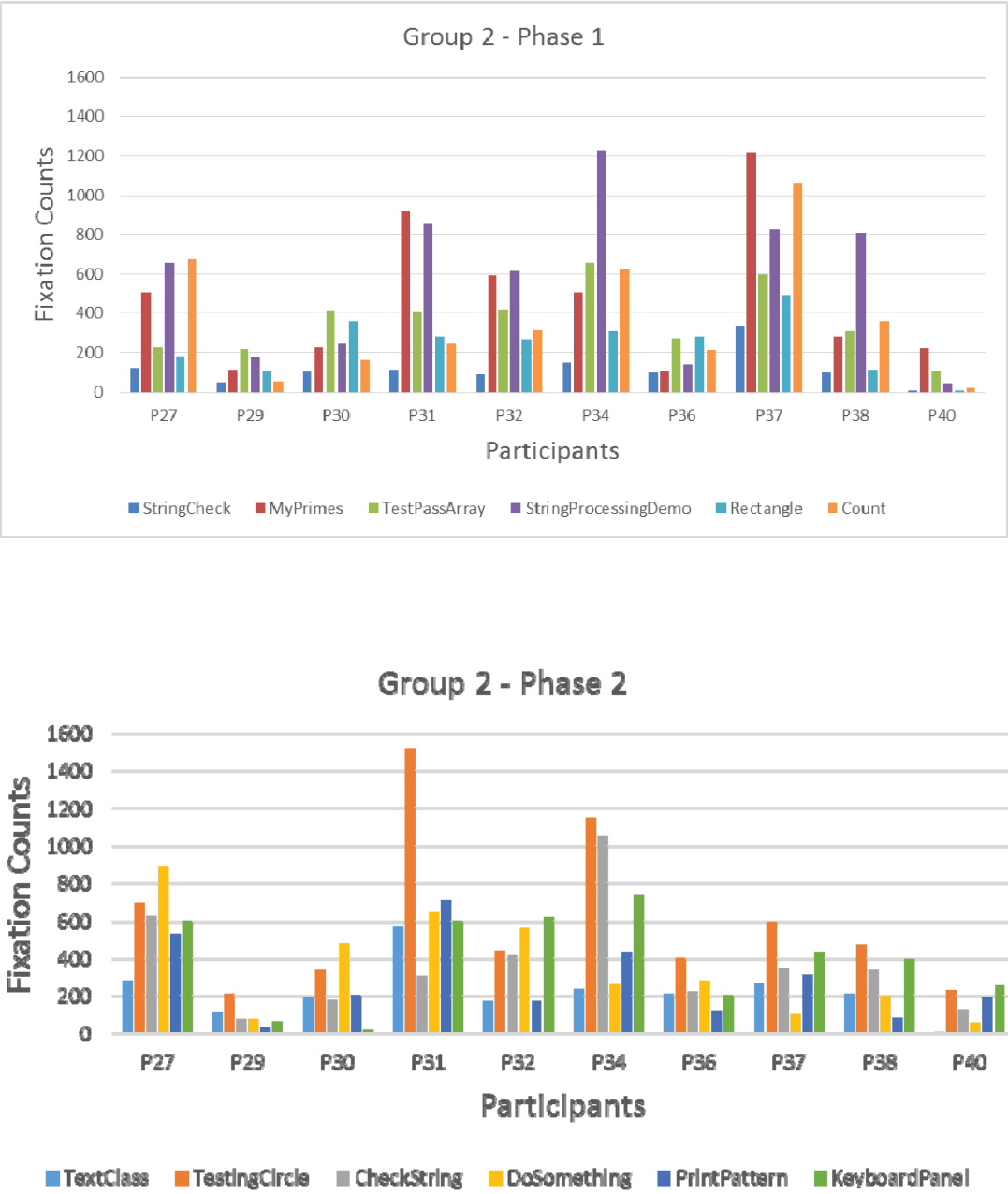
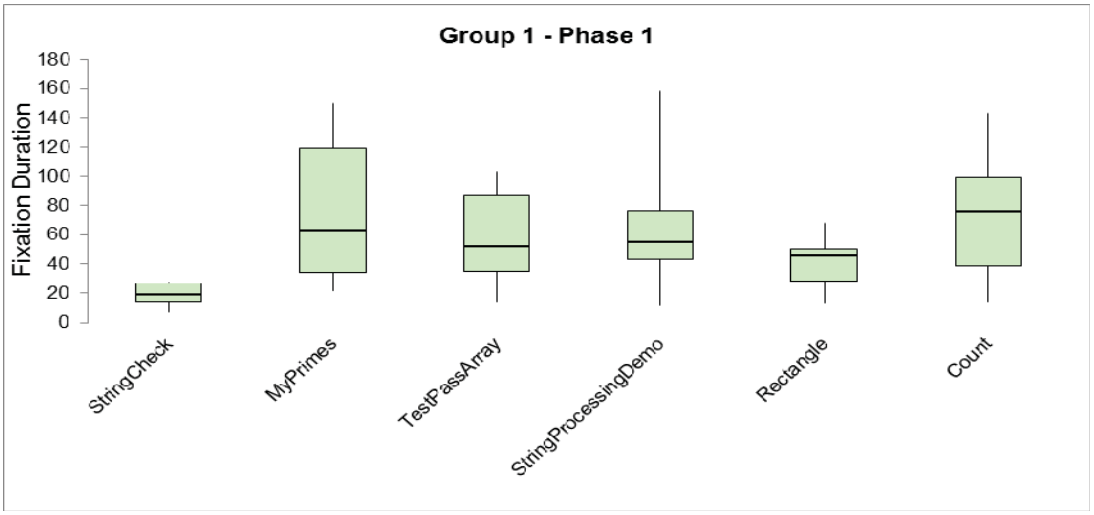


Figure 17. Fixation Counts for Group 2 both Phases

4.5 Fixation Durations

The fixation durations denotes the durations of all the fixations within the AOIs for each program. For example, if a program has 10 lines, we would total the fixation durations for all the fixation counts for each of those 10 lines. This resulting number would give us the fixation duration for that particular program.



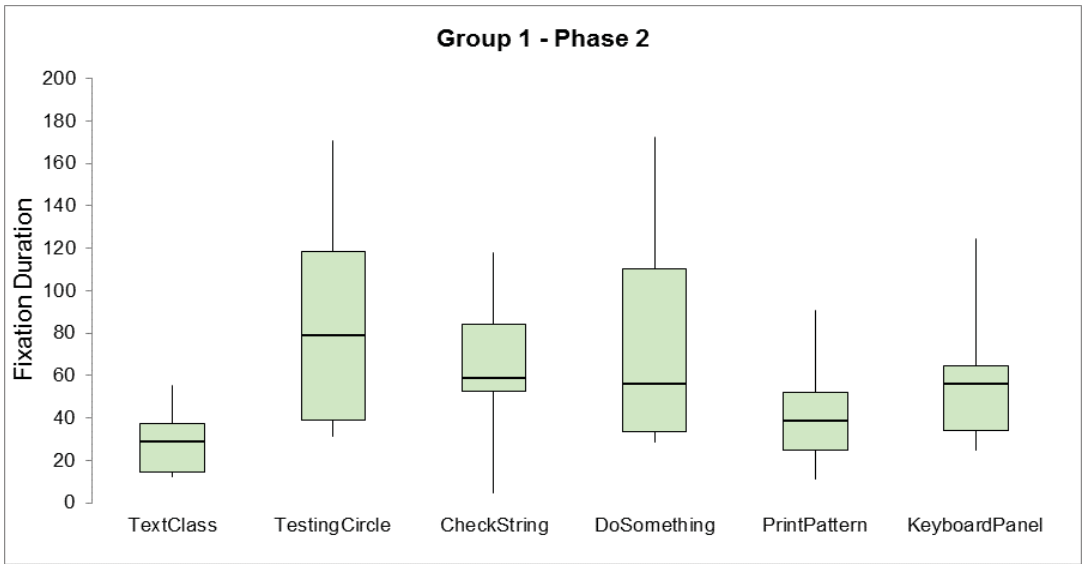
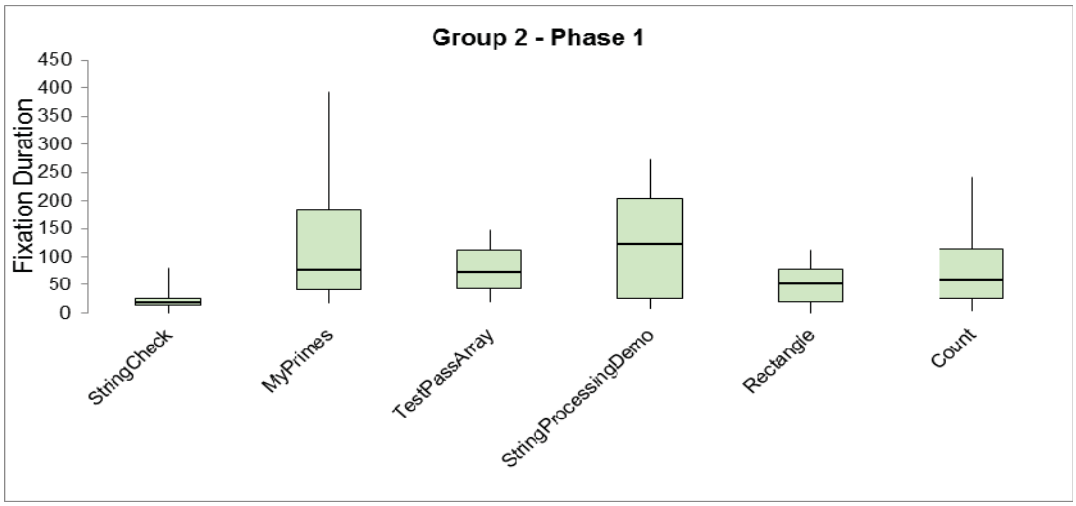


Figure 18. Fixation Duration for Group 1 both Phases



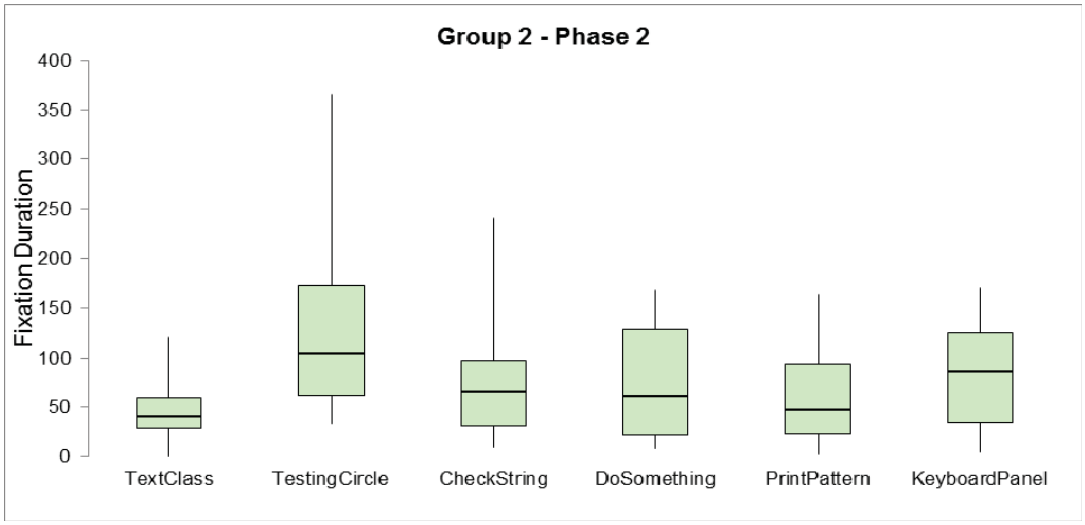


Figure 19. Fixation Duration for Group 2 both Phases

In Group 1, the fixation durations were higher for My 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 MyPrimes 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.

4.6 Correlation: Fixation Count vs. Fixation Duration

We now discuss any relationship, if one exists, between the two dependent variables: fixation count and fixation duration. Refer to Figure 20 and Figure 21 for the scatterplots comparing these two dimensions for all the programs in total. We only report this for Group 1 since Group 2's graphs are similar.

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 Figure 21. The slope of both these graphs is not the same indicating that in Phase 2, novices took the study more seriously taking longer time to accurately answer the questions.

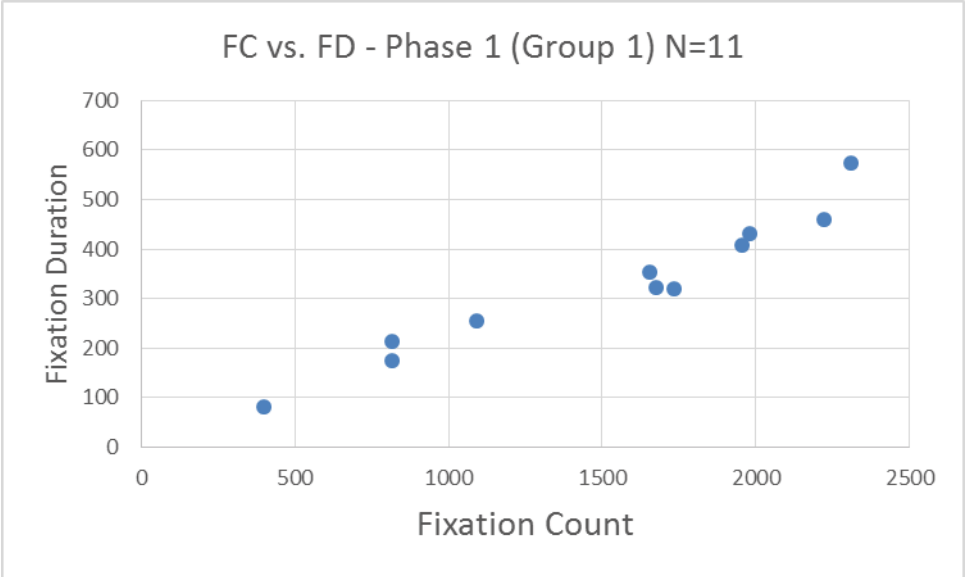


Figure 20. Fixation Count vs. Fixation Duration for Group 1 Phase 1

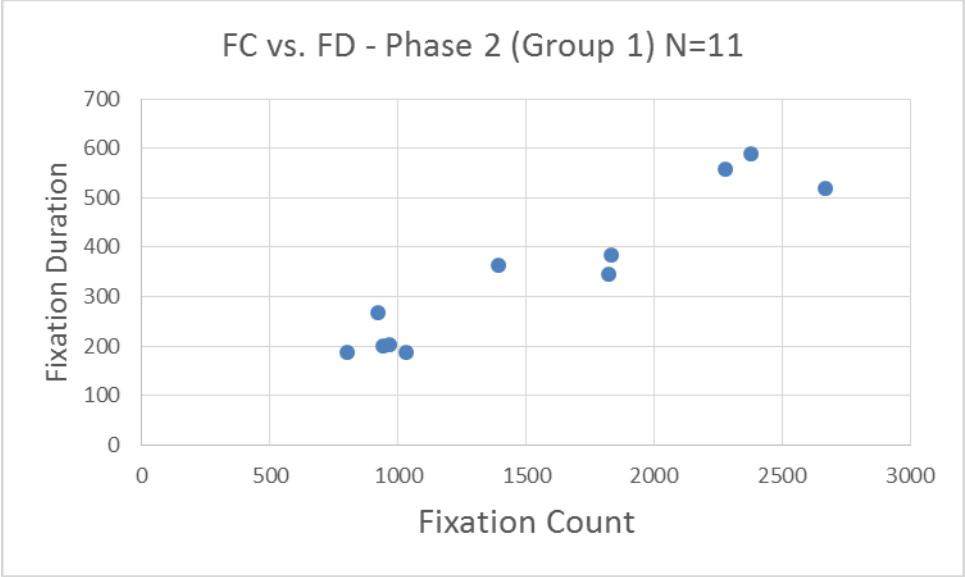


Figure 21. Fixation Count vs. Fixation Duration for Group 1 Phase 2

4.7 Correlation: Time vs. Accuracy

In this section, we report on the correlation between time and accuracy. The two figures below show 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.

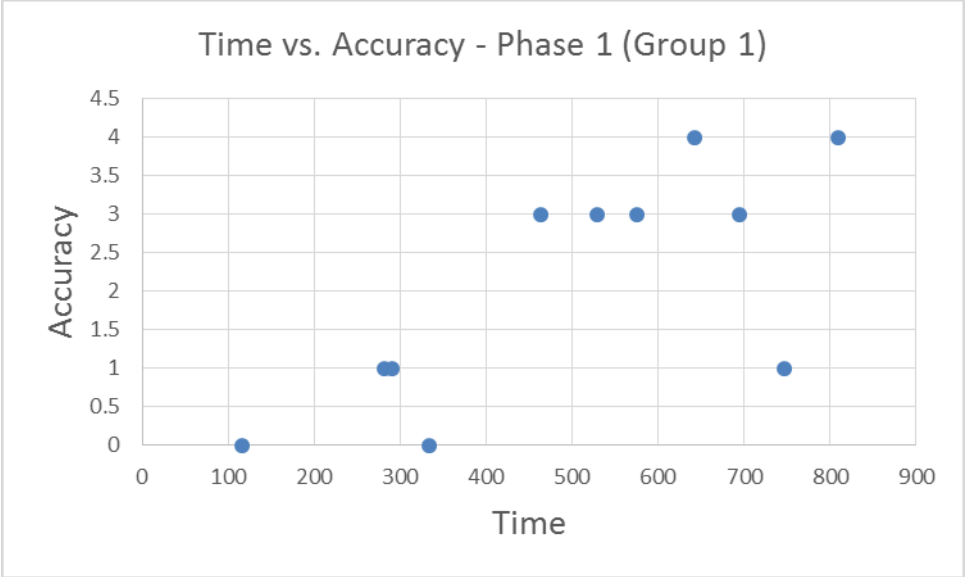


Figure 22. Time vs. Accuracy for Group 1 Phase 1

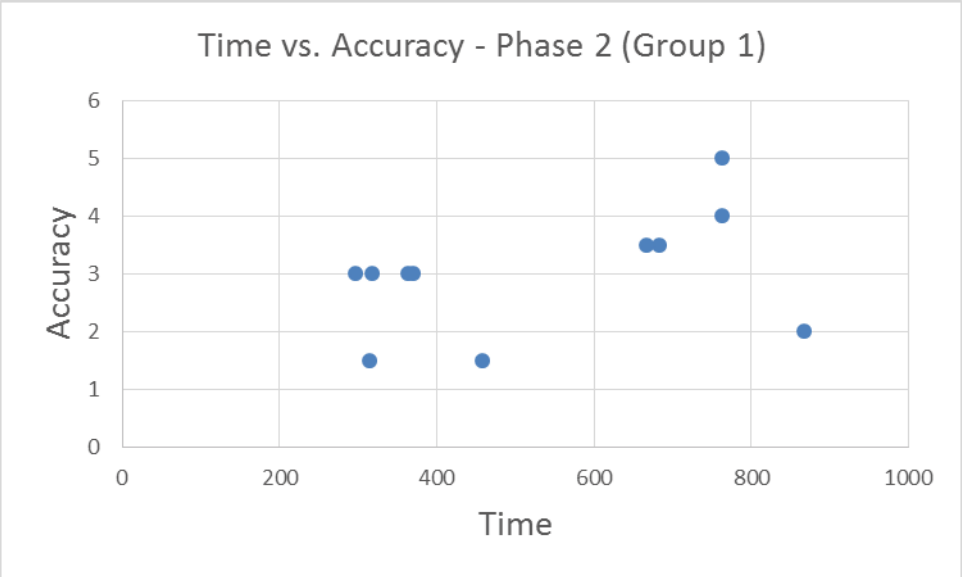


Figure 23. Time vs. Accuracy for Group 1 Phase 2

4.8 Comparing Phase 1 and Phase 2

In this section, we compare phase 1 with phase 2 on several dimensions including the three individual programs that were tested across both phases to determine if any learning occurred in those specific areas.

4.8.1 Accuracy and Time Comparison between Phases

The accuracy and time comparisons for the phases in each group are presented in Figure 24 and Figure 25. These are cumulative totals across all tasks for each phase. From Figure 24, we observe that the accuracy was higher overall for both groups, with the second group performing about half a point better than group 1. From Figure 25 we observe that the time taken by group 1 was mostly the same across both phases. However, we did notice that the minimum time increased in the second phase indicating

that more students put in the effort to read and try to partially comprehend the program. In Group 2, the time mostly remained the same overall. The maximum value of the distribution was less in Phase 2 however, when compared to Phase 1.

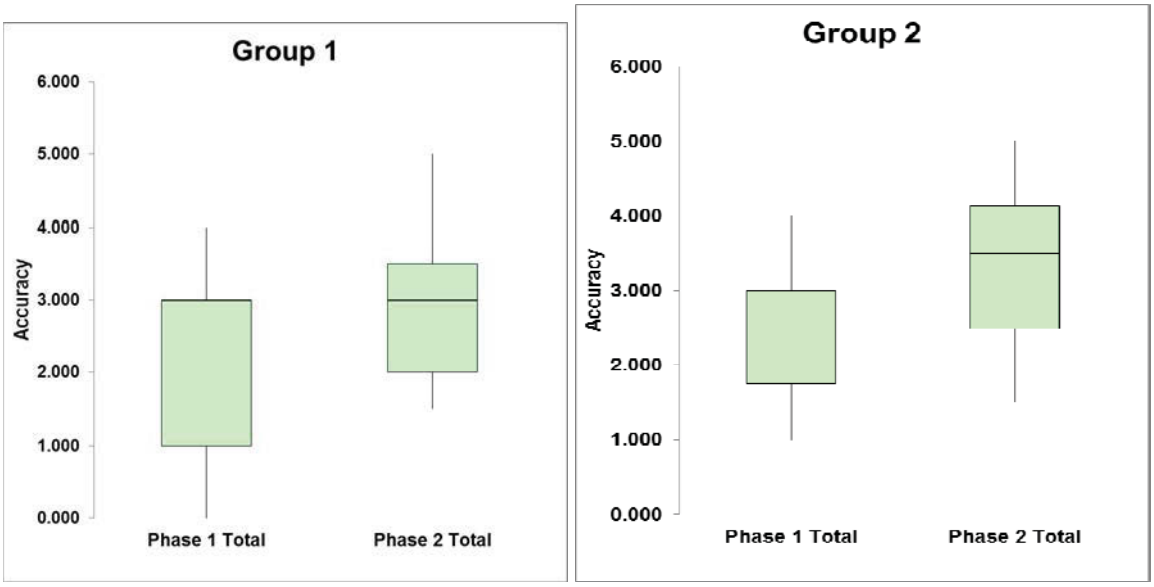


Figure 24. Accuracy for both groups in both phases

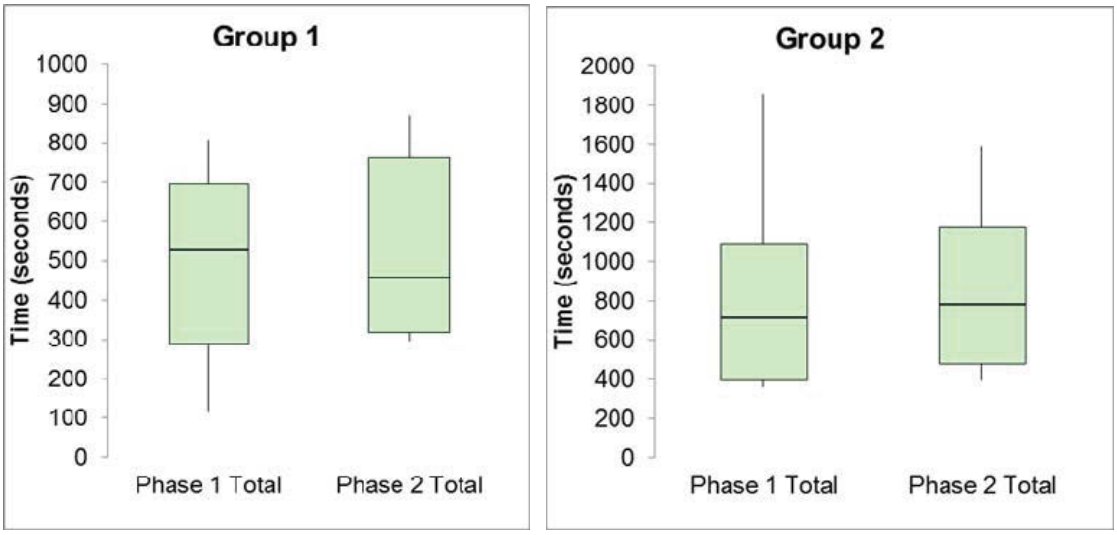


Figure 25. Time for both groups in both phases

4.8.2 Accuracy and Time Comparison for Three Program Pairs in Group 1

We present a comparison of three programs in the next two figures. The program Count is matched with PrintPattern, Primes is matched with CheckString, and StringProcessingDemo is matched with TextClass. The breakdown is also shown per participant to see more fine-grained differences. We see 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 studetns spent more time reading StringProcessingDemo of Phase 1 than TextClass of Phase 2.

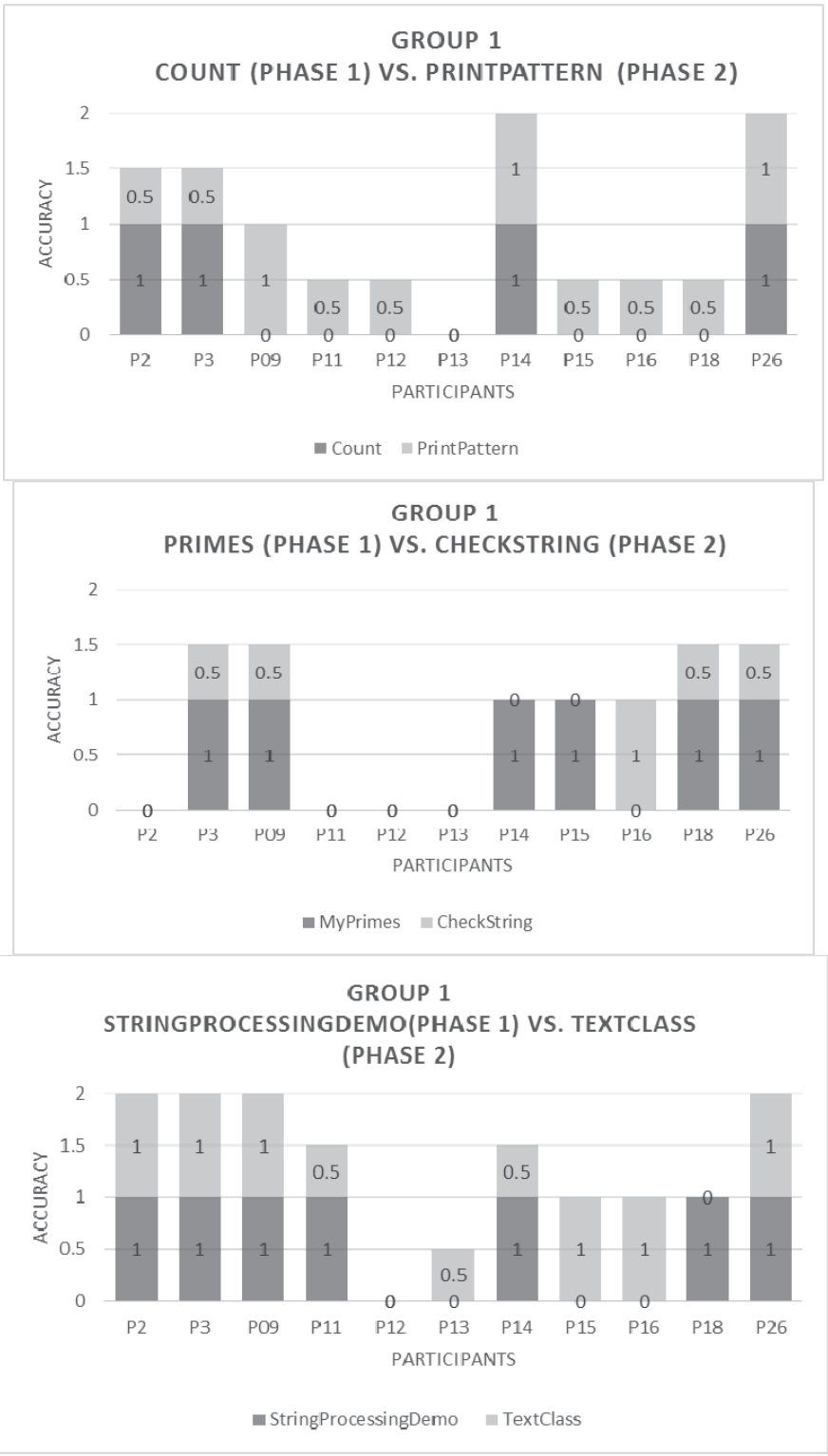
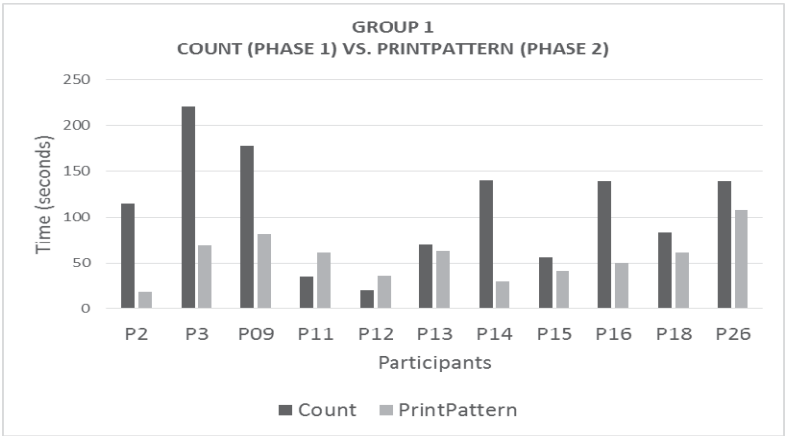
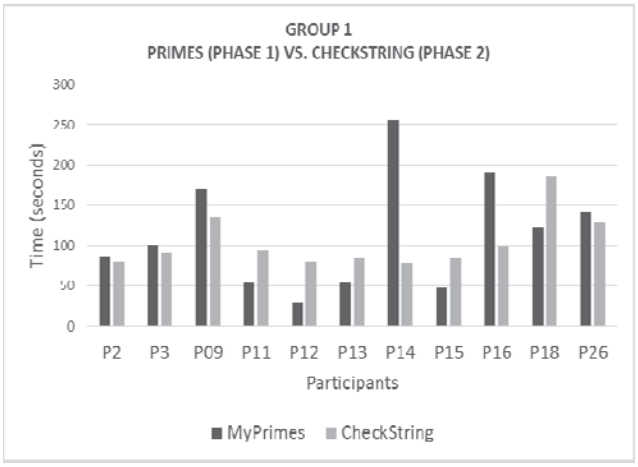


Figure 26. Comparison of three programs in Group 1 for phase 1 and phase 2: Accuracy





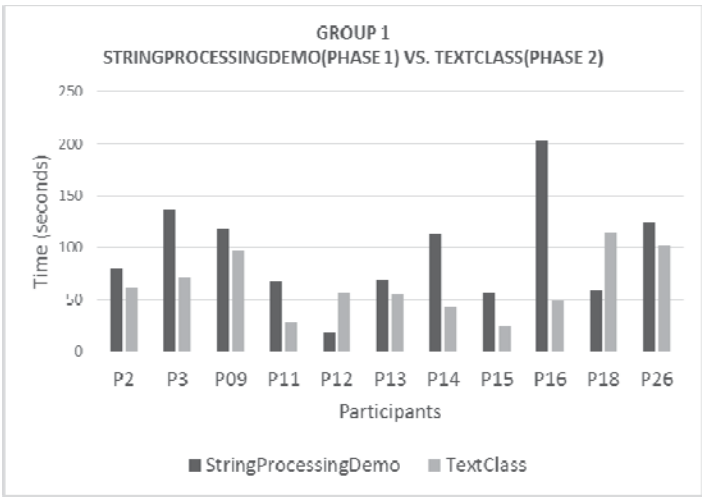


Figure 27. Comparison of three programs in Group 1 for phase 1 and phase 2: Time

4.8.3 Accuracy and Time Comparison for Three Program Pairs in Group 2

This section is similar to the previous sub-section but is related to 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.

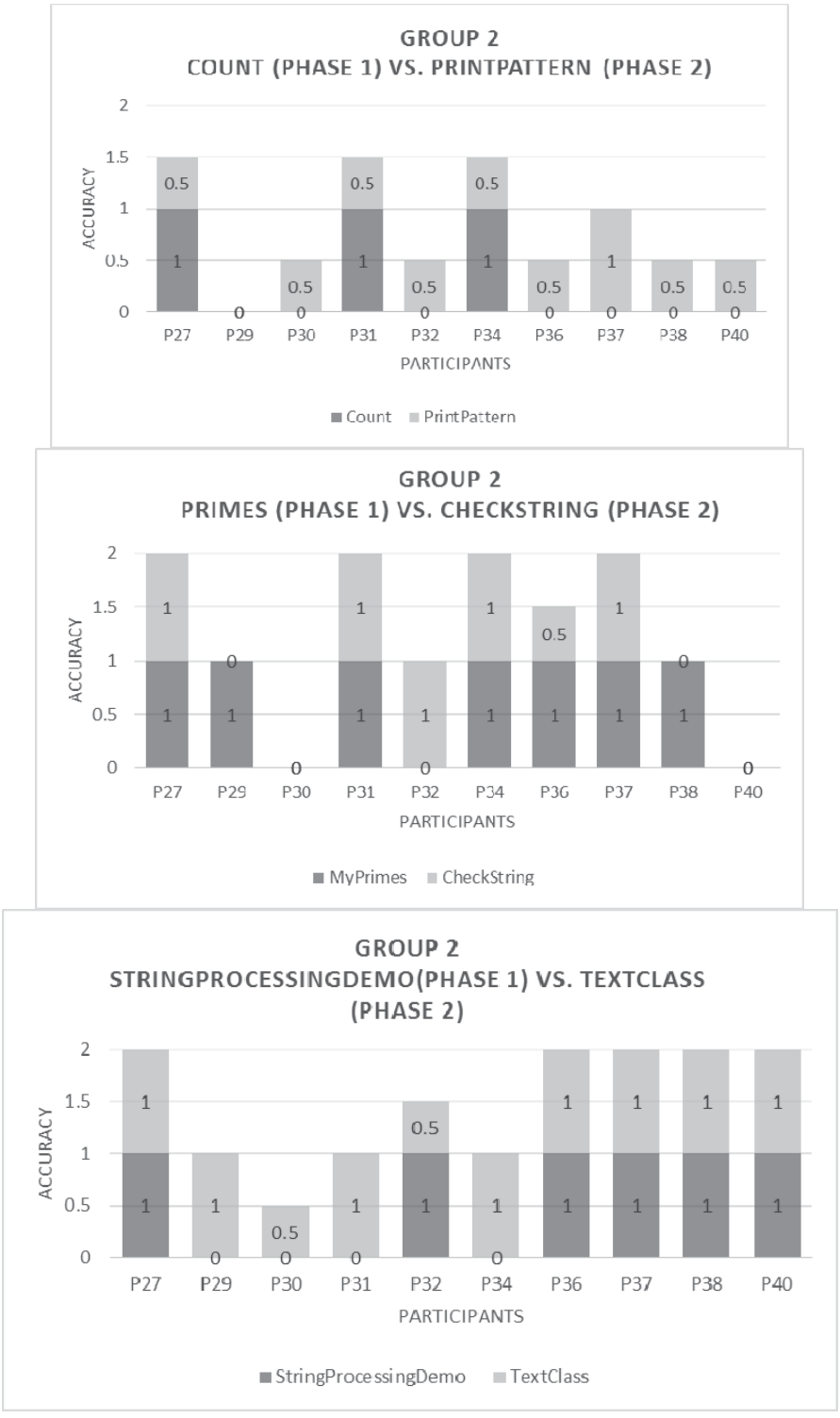
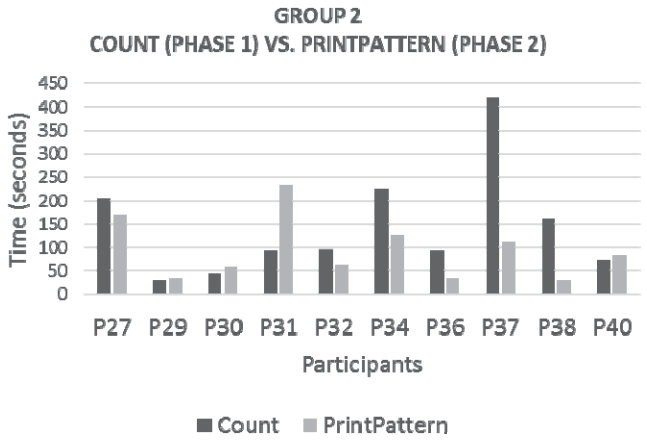
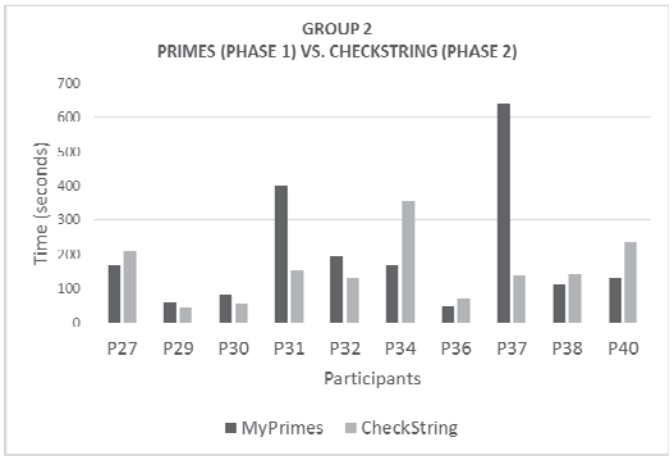


Figure 28. Comparison of three programs in Group 2 for phase 1 and phase 2: Accuracy





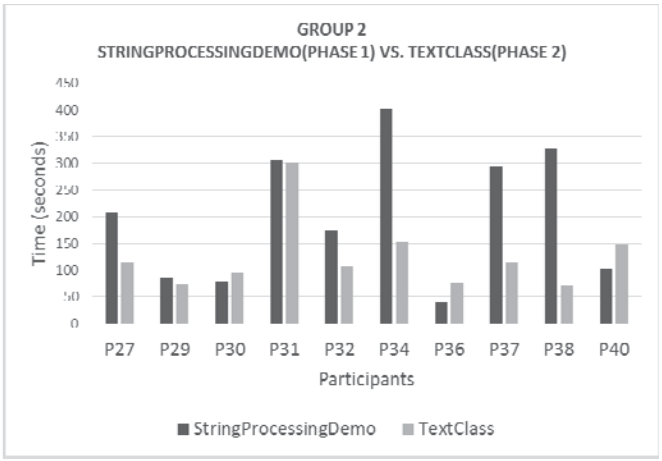


Figure 29. Comparison of three programs in Group 2 for phase 1 and phase 2: Time

4.9 Post Questionnaire Results

Sufficient time was given to all students for completing the tasks, hence none of them complained about this part in the post questionnaire. We now present specific results from each group's post questionnaire.

Group 1 Phase 1:

Out of all the participants none of them felt the programs were easy. Majority of them rated the level of difficulty as moderate, few of them felt very difficult while answering comprehension. The overall view of feedback goes like this. All students had sufficient time to complete programs, Couple of students could not understand few lines of code, It was hard for few students to remember the output, since they can not go back once they proceed to answer comprehension. Sitting still was hard for one student.

Group 1 Phase 2:

Just as in Phase 1, few students felt they needed more time to understand the source code, but eventually did. To mention particular difficulties, one of the student has difficulty in understanding programs calling packages, for another student it was hard to understand few functions, one of the student did not understand what key listeners were doing in the program. For another student replacing words in the substring was difficult, and they did not understand exception handling.

Group 2 Phase 1:

Most of the students felt that the source code given was a little familiar, but the main problem was to remember the whole program to write a summary. Some of the

students wanted a pen and paper. Sitting patiently, without moving was also an issue for one student.

Group 2 Phase 2:

One of the students felt that he/she should have brushed up their skills on few topics, another subject had difficulty remembering few concepts, Level of difficulty was moderate for almost all of them except for one student, who felt Phase 2 was harder than Phase 1.

4.10 Discussion

Based on the analysis presented, we find that novices tend to read source code like reading natural language text. They do it in a linear sequential fashion. 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 mental structure occurs.

Refer to the heatmap below showing the eye gaze of one particular novice. It can be seen how the most difficult part that they focused on the most and had trouble with was the ternary operator. A lot of fixations in one single area and in addition repeated fixations over the area (such as regressions) indicate high cognitive load which indicates that the novice is having a hard time trying to understand what the program is doing.

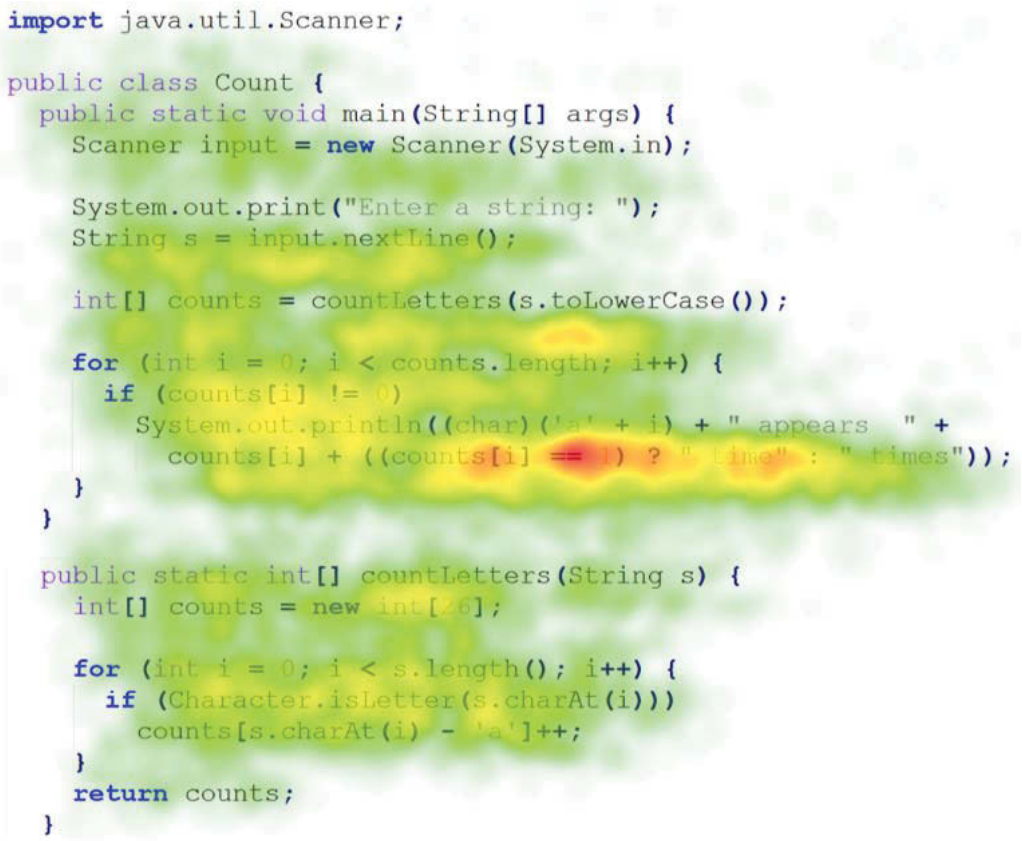


Figure 30. Heatmap of where a novice is having problem.

We now revisit our hypotheses as presented in Section 3.2. We are able to reject H_a and H_t when we take into account all the tasks together and compare Phase 1 and Phase 2 as a whole. However, at this time, we did not find any significant differences between fixation count and duration 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 finegrained across and between lines in the programs. We have left this as a future exercise.

4.11 Threats to Validity

Every experiment is subject to various threats to validity. We discuss some of these now. 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. 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.

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. Most students stated the output of the program in the summary.

CHAPTER 5

Conclusions and Future Work

Computer Science is currently being taught at most major Universities with a focus on code writing without really introducing methods on how to first read the code. Reading code is important because it is the first thing developers do as part of most software tasks such as bug fixing and impact analysis. How does a novice read and comprehend code? In order 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 by having a 7-week separation between the two 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 different in terms of time. In fact, more time was spent in Phase 2 to produce a correct program. This was a surprise to us but it does make sense. Students were trying hard to understand code and put in more effort even though they clearly found the tasks difficult. Group 2 that 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, the best way to determine when a novice becomes an expert would be to follow a small sample of students across their undergraduate study and determine via their eye movements, the time when chunking becomes obvious. When this occurs, we could say that a shift in a development stage has occurred.

APPENDIX Study Material

You will find all the tasks here including the pre and post questionnaires in order to make replicate easier.

A.1. Study Instructions

The purpose of this study is to understand how people comprehend code. It is not to measure your skills at programming.

- You will be given six source code snippets with one comprehension question asked after each code snippet.
- You may study the code snippet for as long as you like.
- Please do not guess the answers.
- The source code given to you does not contain any bugs.
- I would like to encourage you to please think aloud. Verbalize your thoughts as this will help us understand your thought process.
- Please answer the questions from the perspective of a person trying to understand and comprehend the code.
- You will fill in your answers in a web form that will pop up automatically.
- For each question, you will be asked to rate the difficulty level you faced and your confidence in the answer you provided. Use the mouse to select the options.
- When you are done with the reading the code on the screen, click the LEFT mouse button only once to advance to the next screen.
- Important note: You will not be able to see the source code while you are answering the question.
- There is no possibility to go back. Please be sure of your choices before you advance to the next screen.

- Please try to maintain your position in the chair while you do the study so that we do not lose the tracking of your eyes. Moving the chair back or moving yourself back in the chair will cause the eye tracker to stop tracking. Small head movements such as looking at the keyboard to type should be fine.
- Find a comfortable position so we can begin.

- We will first begin with calibrating your eyes. Look at the black dot in the center of the red circle and follow it around on the screen. m
- Then, we will begin with a sample task followed by the six actual study questions

We request that you please not let students who have not taken the study in your class know of the questions asked after you leave. Thank you for maintaining the integrity of the data.

A.2. Background Questionnaire

ID: _____

1. Native Language: _____
2. Your English level?
    a. Non
    b. Low
    c. Medium
    d. High
3. Your overall programming expertise?
    a. None
    b. Low
    c. Medium
    d. High
4. Your overall expertise in Java?
    a. None
    b. Low
    c. Medium
    d. High
5. Age:
    a. < 18 years
    b. 18 – 22 years
    c. 23 – 27 years
    d. > 27 years
6. Gender:
    a. Male
    b. Female
7. Years of experience programming in Java?
    a. None
    b. < 1
    c. 1-2 years
    d. 3-5 years
    e. >5 years
8. How often do you program in Java?
    a. less than 1 hour / month
    b. less than 1 hour / week
    c. less than 1 hour / day
    d. more than 1 hours/day

9. Years of experience programming in any language?
    a. None
    b. < 1
    c. 1-2 years
    d. 3-5 years
    e. >5 years
10. How often do you program in a language other than Java?
    a. less than 1 hour / month
    b. less than 1 hour / week
    c. less than 1 hour / day
    d. more than 1 hours/day
11. When did you first start programming? (list the year): _____
12. Which language did you first learn programming in? _____
13. Which programming languages do you know? State the language and level of expertise (low, medium, high)

    _____
14. When did you take the class CSIS 2610?. If you did not take it enter NO.

    _____
15. When did you take the class CSIS 2605? If yes, approximate month and year else enter NO. _____
16. Do you want to say anything else about your programming background? If yes, you may enter it below.

A.3. Phase 1 Questionnaire

ID:_____

We will begin with a sample task to familiarize you with the process. You will first see Java source code followed by the comprehension questions and the corresponding answer (since this is a sample task).

The answer is indicative of how you should be answering the questions.

You will be shown a short Java program. You will need to study it for as long as you need. Click the LEFT mouse button to continue.

The actual study will take place after the sample.

Say "Begin" to start

Once you go to the next page you cannot go back.

```java
import java.util.Scanner;

class OddOrEven
{
   public static void main(String args[])
   {
      int x;
      x = 15;

      if ( x % 2 == 0 )
         System.out.println("You entered an even number.");
      else
         System.out.println("You entered an odd number.");
   }
}
```

Two possible comprehension questions you could be asked are:

What is the output of the program?
OR
Give a summary of the program.

Had this been the actual task, you would have written your answers in space provided for you on the next screen.

Since this is the sample, the next slide shows you what is expected as answers for this sample task.

```java
import java.util.Scanner;

class OddOrEven
{
   public static void main(String args[])
   {
      int x;
      x = 15;

      if ( x % 2 == 0 )
         System.out.println("You entered an even number.");
      else
         System.out.println("You entered an odd number.");
   }
}
```

What is the output?

You entered an odd number

Summary

The program checks if the number 15 is odd or even. It displays that you entered an even number or odd number depending on the value of x. In this case, it is an odd number.

We will now begin the actual study. You will be shown six code segments.

You may study each code segment for as long as you like. Click on the LEFT mouse button to proceed to the next screen.

After each code segment you will be asked one question.

Important: You will not be able to view the code while you are answering the question.

After you are done reading the code you will have space provided to write your answer.

After each question you will also fill in the confidence and difficulty ratings for the question you just answered.

Note that once you LEFTCLICK to move forward you will not be able to go back.

```java
public class StringCheck
{
    public static void main(String[] args)
    {
        String test2 = new String("Java isn't just for breakfast.");
        String t2 = new String("Java isn't just for breakfast.");

        if (test2 == t2)
            System.out.println("SAME");
        else
            System.out.println("DIFFERENT");
    }
}
```

What is the output of StringCheck.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
import java.util.Scanner;

public class MyPrimes {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Find all prime numbers <= n, enter n: ");
    int n = input.nextInt();

    boolean[] primes = new boolean[n + 1];

    for (int i = 0; i < primes.length; i++) {
      primes[i] = true;
    }

    for (int k = 2; k <= n / k; k++) {
      if (primes[k]) {
        for (int i = k; i <= n / k; i++) {
          primes[k * i] = false;
        }
      }
    }

    final int NUMBER_PER_LINE = 10;
    int count = 0;

    for (int i = 2; i < primes.length; i++) {
      if (primes[i]) {
        count++;
        if (count % 10 == 0)
          System.out.printf("%7d\n", i);
        else
          System.out.printf("%7d", i);
      }
    }
    System.out.println("\n" + count + " number of primes <= " + n);
  }
}
```

Write a summary for MyPrimes.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
public class TestPassArray {
  public static void main(String[] args) {
    int[] a = {1, 2};

    System.out.println("Before invoking swap");
    System.out.println("array is [" + a[0] + ", " + a[1] + "]");
    swap(a[0], a[1]);
    System.out.println("After invoking swap");
    System.out.println("array is [" + a[0] + ", " + a[1] + "]");

    System.out.println("Before invoking swapFirstTwoInArray");
    System.out.println("array is [" + a[0] + ", " + a[1] + "]");
    swapFirstTwoInArray(a);
    System.out.println("After invoking swapFirstTwoInArray");
    System.out.println("array is [" + a[0] + ", " + a[1] + "]");
  }
  public static void swap(int n1, int n2) {
    int temp = n1;
    n1 = n2;
    n2 = temp;
  }

  public static void swapFirstTwoInArray(int[] array) {
    int temp = array[0];
    array[0] = array[1];
    array[1] = temp;
  }
}
```

What is the output of TestPassArray.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```
public class StringProcessingDemo
{
    public static void main(String[] args)
    {
        String sentence = "I hate text processing!";
        int position = sentence.indexOf("hate");
        String ending = sentence.substring(position + "hate".length( ));

        System.out.println("012345678901234567890123");
        System.out.println(sentence);
        System.out.println("The word \"hate\" starts at index " + position);

        sentence = sentence.substring(0, position) + "adore" + ending;
        System.out.println("The changed string is:");
        System.out.println(sentence);
    }
}
```

Write a summary for StringProcessingDemo.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
public class Rectangle {
    private int x1, y1, x2, y2;

    public Rectangle (int x1, int y1, int x2, int y2) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }

    public int width() {
        return this.x2 - this.x1;
    }

    public int height() {
        return this.y2 - this.y1;
    }

    public double area() {
        return this.width() * this.height();
    }

    public static void main(String[] args) {
        Rectangle rect1 = new Rectangle(0, 0, 10, 10);
        System.out.println(rect1.area());

        Rectangle rect2 = new Rectangle(5, 5, 10, 10);
        System.out.println(rect2.area());
    }
}
```

What is the output of Rectangle.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
import java.util.Scanner;

public class Count {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);

    System.out.print("Enter a string: ");
    String s = input.nextLine();

    int[] counts = countLetters(s.toLowerCase());

    for (int i = 0; i < counts.length; i++) {
      if (counts[i] != 0)
        System.out.println((char)('a' + i) + " appears  " +
          counts[i] + ((counts[i] == 1) ? " time" : " times"));
    }
  }

  public static int[] countLetters(String s) {
    int[] counts = new int[26];

    for (int i = 0; i < s.length(); i++) {
      if (Character.isLetter(s.charAt(i)))
        counts[s.charAt(i) - 'a']++;
    }
    return counts;
  }
}
```

Write a summary for Count.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

A.4. Phase 2 Questionnaire

We will begin with a sample task to familiarize you with the process.

You will be shown a short Java program. You will need to study it for as long as you need. Click the LEFT mouse button to continue.

Then you will be asked to summarize the program. Please try to be as accurate as possible when you describe the program. Please do not guess the answer.

In the sample task, you will see the answers so you know the method we expect you to answer the actual tasks.

The actual study will take place after the sample.

Say "Begin" to start with the sample task.

Once you click and go to the next page you cannot go back.

```java
import java.util.Scanner;

class OddOrEven
{
   public static void main(String args[])
   {
      int x;
      x = 15;

      if ( x % 2 == 0 )
         System.out.println("You entered an even number.");
      else
         System.out.println("You entered an odd number.");
   }
}
```

You will be asked to

Give a summary of the program you read.

Please do not guess the answer.

On the next slide you will see an example of the summary of the program you just read.

```java
import java.util.Scanner;

class OddOrEven
{
   public static void main(String args[])
   {
      int x;
      x = 15;

      if ( x % 2 == 0 )
         System.out.println("You entered an even number.");
      else
         System.out.println("You entered an odd number.");
   }
}
```

What is the output?

You entered an odd number

Summary

The program checks if the number 15 is odd or even. It displays that you entered an even number or odd number depending on the value of x. In this case, it is an odd number.

We will now begin the actual study. You will be shown six code segments.

You may study each code segment for as long as you like. Click on the LEFT mouse button to proceed to the next screen.

After each code segment you will be asked to summarize the program.

Important: You will not be able to view the code while you are summarizing the program.

After you are done reading the code you will have space provided to write your answer.

After each question you will also fill in the confidence and difficulty ratings for the question you just answered.

Note that once you LEFTCLICK to move forward you will not be able to go back.

```java
public class TextClass {
    public static void main ( String [ ] args ) {
        String text = "Hello World!" ;

        int positionW = text.indexOf( "W") ;
        int textLength = text.length () ;

        String word = text.substring ( positionW , textLength - 1 );

        System.out.print ( text.replace ( word , "Hello" ) ) ;
    }
}
```

Write a summary for TextClass.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```
public class TestingCircle {
  public static void main(String[ ] args) {
    try {
      Circle c1 = new Circle( 5 );
      c1.setRadius( -5 );
      Circle c3 = new Circle( 30 );
    }
    catch ( InvalidRadiusException ex ) {
      System.out.println ( ex );
    }
    System.out.println("Number of objects created: " +
    Circle.getNumberOfObjects( ) );
  }
}

class InvalidRadiusException extends Exception {
  private double radius;
  public InvalidRadiusException ( double radius ) {
    super ("Invalid radius " + radius);
    this.radius = radius;
  }
  public double getRadius ( ) {
    return radius;
  }
}
```

```
class Circle {
  private double radius;
  private static int numberOfObjects = 0;
  public Circle ( ) {
    this ( 1.0 );
  }
  public Circle ( double newRadius ) {
    try {
      setRadius ( newRadius );
      numberOfObjects++;
    }
    catch ( InvalidRadiusException ex ) {
      ex.printStackTrace( );
    }
  }

  public void setRadius ( double newRadius )
                              throws InvalidRadiusException {
    if ( newRadius >= 0)
      radius =  newRadius;
    else
      throw new InvalidRadiusException ( newRadius );
  }

  public static int getNumberOfObjects ( ) {
    return numberOfObjects;
  }

  public double findArea ( ) {
    return radius * radius * 3.14159;
  }
}
```

Write a summary for TestingCircle.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
import java.util.Scanner;

public class CheckString {
  public static void main (String[] args) {
    Scanner input = new Scanner ( System.in );
    System.out.print ("Enter a string: ");
    String s = input.nextLine();
    if (isSomething ( s ))
      System.out.println (s + " is a ___________");
    else
      System.out.println (s + " is not a _______________");
  }

  public static boolean isSomething ( String s ) {
    int low = 0;
    int high = s.length() - 1;

    while ( low < high ) {
      if (s.charAt( low ) != s.charAt( high ))
        return false;

      low++;
      high--;
    }
    return true;
  }
}
```

Write a summary for CheckString.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
public class DoSomething {

  public static void somethingUseful ( double [ ] list ) {
    for (int i = list.length - 1  ;  i >= 1  ;  i--) {
      double currentMax  =  list[ 0 ];
      int currentMaxIndex  =  0;

      for (int j = 1  ;  j <= i  ;  j++) {
        if (currentMax  <  list[ j ]) {
          currentMax  =  list[ j ];
          currentMaxIndex  =  j;
        }
      }

      if (currentMaxIndex != i) {
        list[ currentMaxIndex ] = list[ i ];
        list[ i ] = currentMax;
      }
    }
  }
}
```

Write a summary for DoSomething.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```java
public class PrintPattern {

  public static void printMethod ( int numberOfRows ) {
      for ( int row = 1  ;  row <= numberOfRows  ;  row ++ ) {
          for ( int col = 1  ;  col <= row  ;  col ++) {
              System.out.print ( '*' );
          }
          System.out.println ( );
      }
  }

  public static void main ( String [ ] args ) {
      PrintPattern.printMethod ( 3 ) ;
  }
}
```

Write a summary for PrintPattern.java?

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
static class KeyboardPanel extends JPanel {
  private int x = 100;
  private int y = 100;
  private char keyChar = 'A';

  public KeyboardPanel( ) {
    addKeyListener ( new KeyAdapter () {
      @Override
      public void keyPressed ( KeyEvent e ) {
        switch ( e.getKeyCode( ) ) {
          case KeyEvent.VK_DOWN: y += 10;   break;
          case KeyEvent.VK_UP: y -= 10;     break;
          case KeyEvent.VK_LEFT: x -= 10;   break;
          case KeyEvent.VK_RIGHT: x += 10;  break;
          default: keyChar = e.getKeyChar();
        }
        repaint();
      }
    });
  }

  @Override
  protected void paintComponent ( Graphics g ) {
    super.paintComponent ( g );
    g.setFont ( new Font ( "TimesRoman", Font.PLAIN, 24 ) );
    g.drawString ( String.valueOf ( keyChar ), x, y);
  }
}
```

Write a summary for KeyboardPanel.java.

How confident are you of your answer?

Low Medium High

How difficult did you think this task was?

Easy Average Difficult

A.5. Post Questionnaire

1. ID: _____

2. Did you have sufficient time to complete the study?
    a. Yes
    b. No

3. How would you rate the overall difficulty of fixing the bugs?
    a. Very easy
    b. Easy
    c. Average
    d. Difficult
    e. Very Difficult

4. Describe any difficulty you faced during the study.

5. Any other comments about the study are welcome.

References

Al Maqbali, Hilal, Falk Scholer, James A. Thom, and Mingfang Wu. 2013. "Using Eye Tracking for Evaluating Web Search Interfaces." In *Proceedings of the 18th Australasian Document Computing Symposium*, 2–9. ADCS '13. New York, NY, USA: ACM. doi:10.1145/2537734.2537747.

Atterer, Richard, Monika Wnuk, and Albrecht Schmidt. 2006. "Knowing the User's Every Move: User Activity Tracking for Website Usability Evaluation and Implicit Interaction." In *Proceedings of the 15th International Conference on World Wide Web*, 203–12. WWW '06. New York, NY, USA: ACM. doi:10.1145/1135777.1135811.

Bartels, Mike, and Sandra P. Marshall. 2006. "Eye Tracking Insights into Cognitive Modeling." In *Proceedings of the 2006 Symposium on Eye Tracking Research &Amp; Applications*, 141–47. ETRA '06. New York, NY, USA: ACM. doi:10.1145/1117309.1117358.

Bernhaupt, Regina, Manfred Eckschlager, and Manfred Tscheligi. 2007. "Methods for Evaluating Games: How to Measure Usability and User Experience in Games?" In *Proceedings of the International Conference on Advances in Computer Entertainment Technology*, 309–10. ACE '07. New York, NY, USA: ACM. doi:10.1145/1255047.1255142.

Binkley, Dave, Marcia Davis, Dawn Lawrie, Jonathan I. Maletic, Christopher Morrell, and Bonita Sharif. 2012. “The Impact of Identifier Style on Effort and Comprehension.” *Empirical Software Engineering* 18 (2): 219–76. doi:10.1007/s10664-012-9201-4.

Busjahn, Teresa, and Carsten Schulte. 2013. “The Use of Code Reading in Teaching Programming.” In *Proceedings of the 13th Koli Calling International Conference on Computing Education Research*, 3–11. Koli Calling ’13. New York, NY, USA: ACM. doi:10.1145/2526968.2526969.

Busjahn, Teresa, Carsten Schulte, and Andreas Busjahn. 2011. “Analysis of Code Reading to Gain More Insight in Program Comprehension.” In *Proceedings of the 11th Koli Calling International Conference on Computing Education Research*, 1–9. Koli Calling ’11. New York, NY, USA: ACM. doi:10.1145/2094131.2094133.

Busjahn, Teresa, Carsten Schulte, Bonita Sharif, Simon, Andrew Begel, Michael Hansen, Roman Bednarik, et al. 2014. “Eye Tracking in Computing Education.” In *Proceedings of the Tenth Annual Conference on International Computing Education Research*, 3–10. ICER ’14. New York, NY, USA: ACM. doi:10.1145/2632320.2632344.

Duchowski, Andrew T. 2007. *Eye Tracking Methodology: Theory and Practice*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Duchowski, Andrew T., Eric Medlin, Anand Gramopadhye, Brian Melloy, and Santosh Nair. 2001. “Binocular Eye Tracking in VR for Visual Inspection Training.” In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 1–8. VRST ’01. New York, NY, USA: ACM. doi:10.1145/505008.505010.

Ehmke, Claudia, and Stephanie Wilson. 2007. “Identifying Web Usability Problems from Eye-Tracking Data.” In *Proceedings of the 21st British HCI Group Annual Conference on People and Computers: HCI...But Not As We Know It - Volume 1*, 119–28. BCS-HCI ’07. Swinton, UK, UK: British Computer Society. http://dl.acm.org/citation.cfm?id=1531294.1531311.

Fan, Quyin. 2010. “The Effects of Beacons, Comments, and Tasks on Program Comprehension Process in Software Maintenance”. Catonsville, {MD}, {USA}: University of Maryland at Baltimore County.

Faro, A., D. Giordano, C. Spampinato, D. De Tommaso, and S. Ullo. 2010. “An Interactive Interface for Remote Administration of Clinical Tests Based on Eye Tracking.” In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, 69–72. ETRA ’10. New York, NY, USA: ACM. doi:10.1145/1743666.1743683.

Guéhéneuc, Yann-Gaël. 2006. “TAUPE: Towards Understanding Program Comprehension.” In *Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*, 1–13. Toronto, Ontario, Canada: ACM.

Hasse, Catrin, Dietrich Grasshoff, and Carmen Bruder. 2012. “How to Measure Monitoring Performance of Pilots and Air Traffic Controllers.” In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 409–12. ETRA ’12. New York, NY, USA: ACM. doi:10.1145/2168556.2168649.

Imants, Puck, and Tjerk de Greef. 2014. “Eye Metrics for Task-Dependent Automation.” In *Proceedings of the 2014 European Conference on Cognitive*

Ergonomics, 23:1–23:4. ECCE '14. New York, NY, USA: ACM. doi:10.1145/2637248.2637274.

Jeanmart, Sebastien, Yann-Gael Gueheneuc, Houari Sahraoui, and Naji Habra. 2009. "Impact of the Visitor Pattern on Program Comprehension and Maintenance." In *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, 69–78. IEEE. http://dl.acm.org/citation.cfm?id=1671248.1671255.

Just, Marcel A., and Patricia A. Carpenter. 1980. "A Theory of Reading: From Eye Fixations to Comprehension." *Psychological Review* 87 (4): 329–54. doi:10.1037/0033-295X.87.4.329.

Kagdi, Huzefa, Shehnaaz Yusuf, and Jonathan I. Maletic. 2007. "On Using Eye Tracking in Empirical Assessment of Software Visualizations." In *Proceedings of the 1st ACM International Workshop on Empirical Assessment of Software Engineering Languages and Technologies: Held in Conjunction with the 22Nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, 21–22. WEASELTech '07. New York, NY, USA: ACM. doi:10.1145/1353673.1353678.

Madsen, Adrian, Adam Larson, Lester Loschky, and N. Sanjay Rebello. 2012. "Using ScanMatch Scores to Understand Differences in Eye Movements Between Correct and Incorrect Solvers on Physics Problems." In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 193–96. ETRA '12. New York, NY, USA: ACM. doi:10.1145/2168556.2168591.

Pfeiffer, Thies, Sophie Stellmach, and Yusuke Sugano. 2014. "4th International Workshop on Pervasive Eye Tracking and Mobile Eye-Based Interaction." In

Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication, 1085–91. UbiComp '14 Adjunct. New York, NY, USA: ACM. doi:10.1145/2638728.2641686.

Rayner, Keith. 1998. "Eye Movements in Reading and Information Processing: 20 Years of Research." *Psychological Bulletin* 124 (3): 372–422. doi:10.1037//0033-2909.124.3.372.

Rosengrant, David. 2010. "Gaze Scribing in Physics Problem Solving." In *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications*, 45–48. ETRA '10. New York, NY, USA: ACM. doi:10.1145/1743666.1743676.

Sadasivan, Sajay, Joel S. Greenstein, Anand K. Gramopadhye, and Andrew T. Duchowski. 2005. "Use of Eye Movements As Feedforward Training for a Synthetic Aircraft Inspection Task." In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 141–49. CHI '05. New York, NY, USA: ACM. doi:10.1145/1054972.1054993.

Sharafi, Zohreh, Zéphyrin Soh, Yann-Gael Gueheneuc, and Giuliano Antoniol. 2012. "Women and Men - Different but Equal: On the Impact of Identifier Style on Source Code Reading." *Program Comprehension (ICPC), 2012 IEEE 20th International Conference on*, 27–36. doi:10.1109/ICPC.2012.6240505.

Sharif, Bonita, and Huzefa Kagdi. 2011. "On the Use of Eye Tracking in Software Traceability." In *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, 67–70. TEFSE '11. New York, NY, USA: ACM. doi:10.1145/1987856.1987872.

Sharif, Bonita, and Jonathan I Maletic. 2010a. “An Eye Tracking Study on the Effects of Layout in Understanding the Role of Design Patterns.” *Software Maintenance (ICSM), 2010 IEEE International Conference on*, 1–10. doi:10.1109/ICSM.2010.5609582.

Sharif, Bonita, and Jonathan I. Maletic. 2010b. “An Eye Tracking Study on camelCase and Under_score Identifier Styles.” In *2010 IEEE 18th International Conference on Program Comprehension*, 196–205. IEEE. doi:10.1109/ICPC.2010.41.

Sjoberg, Dag I. K., Tore Dyba, and Magne Jorgensen. 2007. “The Future of Empirical Methods in Software Engineering Research.” In , 358–78. FOSE ’07. Washington, DC, USA: IEEE Computer Society. doi:10.1109/FOSE.2007.30.

Turner, Jayson, Eduardo Velloso, Hans Gellersen, and Veronica Sundstedt. 2014. “EyePlay: Applications for Gaze in Games.” In *Proceedings of the First ACM SIGCHI Annual Symposium on Computer-Human Interaction in Play*, 465–68. CHI PLAY ’14. New York, NY, USA: ACM. doi:10.1145/2658537.2659016.

Turner, Rachel, Michael Falcone, Bonita Sharif, and Alina Lazar. 2014. “An Eye-Tracking Study Assessing the Comprehension of C++ and Python Source Code.” In *Proceedings of the Symposium on Eye Tracking Research and Applications*, 231–34. ETRA ’14. New York, NY, USA: ACM. doi:10.1145/2578153.2578218.

Yusuf, Shehnaaz, Huzefa Kagdi, and Jonathan I. Maletic. 2007. “Assessing the Comprehension of UML Class Diagrams via Eye Tracking.” In *Proceedings of the 15th IEEE International Conference on Program Comprehension*, 113–22.

Youngstown
STATE UNIVERSITY

One University Plaza, Youngstown, Ohio 44555
Office of Grants and Sponsored Programs
330.941.2377
Fax 330.941.1580

September 15, 2014

Dr. Bonita Sharif, Principal Investigator
Mr. Leela Krishna Yenigalla, Co-investigator
Department of Computer Science and Information Systems
UNIVERSITY

RE: IRB Protocol Number: 032-2015
Title: How Novices Read Source Code

Dear Dr. Sharif and Mr. Yenigalla

The Institutional Review Board of Youngstown State University has reviewed the aforementioned Protocol via expedited review, and it is approved with the following conditions:

(1) *Place the name of the student investigator, Leela Yenigalla, on the Informed Consent Form;*
(2) *Use only publically available email addresses without permission.*

Any changes in your research activity should be promptly reported to the Institutional Review Board and may not be initiated without IRB approval except where necessary to eliminate hazard to human subjects. Any unanticipated problems involving risks to subjects should also be promptly reported to the IRB. Best wishes in the conduct of your study.

Sincerely,

[redacted]

Dr. Scott Martin
Interim Associate Dean for Research
Authorized Institutional Official

SCM:cc

c: Dr. Kriss Schueller, Chair
Department of Computer Science and Information Systems