2022 IEEE Frontiers in Education Conference (FIE) | 978-1-6654-6244-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/FIE56618.2022.9962681

Using AI-based NiCATS System to Evaluate Student Comprehension in Introductory Computer Programming Courses

Bradley Boswell
Computer Science
Georgia Southern University
Statesboro, USA
bb05758@georgiasouthern.edu

Andrew Sanders
Computer Science
Georgia Southern University
Statesboro, USA
as13770@georgiasouthern.edu

Andrew Allen
Computer Science
Georgia Southern University
Statesboro, USA
andrewallen@georgiasouthern.edu

Gursimran Singh Walia
Computer Science
Augusta University
Augusta, USA
gwalia@augusta.edu

Md Shakil Hossain
Computer Science
Georgia Southern University
Statesboro, USA
mh34922@georgiasouthern.edu

***Abstract*—This Research to Practice Full Paper presents the use of data collected by our Non-Intrusive Classroom Attention Tracking System (NiCATS) to evaluate student comprehension. Quantifying students' cognitive processes in classrooms in a non-intrusive way is challenging. By analyzing various aspects of the eye metrics against defined regions of interest (ROI), instructors can better understand students' cognitive processes as they acquire new knowledge. Eye-tracking studies primarily define ROIs based on commonly used metrics (source code complexity, significant fixation durations, etc.). While helpful, these metrics, when used independently, do not accurately represent their comprehension patterns. This paper contributes an alternative, multilayered approach for calculating gaze metrics against automatically defined ROIs. The work utilizes the AI-based Non-Intrusive Classroom Attention Tracking System (NiCATS - developed by the researchers), collecting raw-gaze data in real-time as information is presented on a computer screen. This paper reports the results of a study in which undergraduate students in a CS programming course were asked to identify defects seeded in Java programs. Each JAVA program included its own unique sets of ROIS defined using two different granularities: lexer-based and line-based. The ROI sets were then used to calculate relevant eye metrics in the context of each ROI layout. The results of the eye metric analysis at specific ROIs w.r.t their code review task provide insights into the cognitive processes students undergo when trying to comprehend new material. Subdividing this region into lexer-based regions, we determined "content topics" students struggled with (e.g., using complex data types) in a specific area. This feedback is valuable to the instructor as it enables the ability to identify hard-to-comprehend content topics post-hoc and gives the ability to validate student learning in the classroom. While this experiment focused on students in introductory programming courses, we intend to conduct experiments in other learning settings where students are expected to read material on a computer screen or solve actual problems. To summarize, the analysis of these eye metrics using more fine-grained ROIs (lexer-based, line-based) as an extension of complexity-based ROIs provides instructors with deeper insights into the cognitive processes used by students when compared to the current state-of-the-art techniques.**

***Keywords*—Gaze Tracking, Knowledge Gain, Code Comprehension**

I. Introduction

One of the open-ended research problems in Computer Science education is to gauge student comprehension in classrooms (either post-hoc or real-time). While this transcends all classrooms, it is a bigger issue in a computer science lab where students are working behind monitors, thus limiting student-teacher visual interactions. Traditional evaluation like test or quiz results can indicate whether students have understood the material or not, but this lacks insight into how they have approached the solution. This makes it difficult for educators to identify the causes behind why students may be misunderstanding the presented information. By identifying these insights and issues early on, instructors would be able to provide better lectures and guidance, especially for struggling students.

Prior researchers have utilized machine learning and computer vision techniques to automatically determine a student's attentiveness but with limited success in terms of providing insight to comprehension [1,2]. Additionally, researchers have reported certain factors related to eye metrics (for example, average fixation duration) that can be used to understand the intent of the person's mind as they perceive and process information presented to them [3,4]. While facial expressions give insights into students' perceived attentiveness from the observer's perspective, eye tracking metrics can add another layer of insights concerning the students' eyes searching and acquiring content that has not been investigated in a wide range of classroom settings. We assert that educators can benefit by having an arsenal of tool sets that can help them identify eye metrics data correlated with students' knowledge comprehension abilities. The advantage of using this approach is the lack of observational effect due to the passive nature of the system while having the similar attention-judging accuracy as domain expert humans. The passive nature of the system is aptly important as it can be applied to online classes that are ubiquitous as a result of the COVID-19 pandemic.

The background section of this paper includes eye-tracking studies related to student comprehension and attentiveness and how these eye metrics can be used to predict attentiveness and other cognitive states of students. The proposed approach uses the Non-Intrusive Classroom Attention Tracking System (NiCATS) [5] for experimental setup and data collection used for comprehension-related analysis. Educators can use the data collected from NiCATS to understand student comprehension. NiCATS is an AI-enabled data collection application to predict student attentiveness using webcam images and gaze data collected from eye trackers. This research focuses on utilizing NiCATS to understand student comprehension and extend the capabilities of the NiCATS system to assist instructors at improving the student experience in their classrooms. We conducted one experiment in a CS lab testing environment to provide insight into the usage of NiCATS for analyzing students' comprehension patterns with automatically generated regions of interest (ROIs). The results indicated that the eye metric and screenshot data collected by NiCATS can provide meaningful information to instructors regarding student comprehension. One notable use case result was the ability to accurately identify the ordering of the regions used by students to iteratively follow the code execution of a for loop. A broader interpretation of the results also indicates that using multiple unique ROI maps is necessary to accurately locate the links between regions of interest as they relate to student comprehension abilities. These insights from NiCATS can certainly help instructors to gain more understanding of student comprehension patterns instead of using the traditional classroom settings

II. Background

The background describes relevant literature on eye tracking studies related to knowledge comprehension and limitation of the existing research.

Researchers frequently use eye trackers to study the cognitive processes related to comprehension from the subject's perspective. Eye tracking is a fast-growing research field, and it has many applications in the measurement of attentiveness, emotion, and cognition. In the context of comprehension studies that use eye tracking hardware, researchers frequently use the following terms:

- Eye gaze data: The immediate direction of a person's eyes translated to (x,y) coordinates when looking at a computer monitor.
- Fixations: The stabilization of the eye on a part of a stimulus for a period of time and are usually around 200-300ms [4].
- Saccades: The quick and continuous eye movement between fixations (~50ms on average) [4].
- Region of Interest (ROI) / Area of Interest (AOI): A specific region or area of the computer monitor identified for any purpose. In comprehension studies, these regions are frequently defined as an (x,y) coordinate pair.

For each of these, generally, an eye tracker is used. An eye tracker can be eye-attached (like a contact lens), optical (reflected infrared), or electric potential (electrodes placed around the eyes). General consumer-grade eye trackers are optical tracking, with some being head-mounted and some being computer monitor-mounted.

Hijazi et al. used a desktop eye tracker and a non-intrusive Heart Rate Variability (HRV) monitor to predict good and bad quality code reviews using artificial intelligence techniques [6]. They collected the biometric data while the subjects reviewed code samples and quantified their review quality based on the number of bugs not detected in the provided code. Their results showed that their tool could predict bad reviews of medium and complex programs with a 75%-87% accuracy.

Rodeghero et al. conducted an empirical study of eye movement patterns for subjects doing source code summarization tasks [7]. The study compared the patterns of professional programmers reviewing source code and found that all 10 subjects followed nearly identical eye movement patterns which were similar to reading natural language. Qualitative findings of this study showed that programmers had the following tendencies: reading code from left to right but not always top to bottom, skimming source code instead of thoroughly reading it, tendency to scan sectionally (transitions between two fixation coordinates are usually within 2 lines of the previously fixated line).

Fritz et al. conducted an experiment with 15 professional programmers where data was collected from an eye tracker, an electrodermal activity sensor, and an electroencephalography sensor which was used to predict whether developers would find a task to be difficult or not [8]. Their classifier was able to predict whether a task would be easy or difficult to a new developer with a 64.99% precision and 64.58% recall.

Veliyath et al. used data collected from self-reporting and an eye tracker as a non-intrusive means to predict student attention over the duration of a class [3]. The researchers mounted eye trackers on the monitors of the computers in a computer lab and had students periodically make note of how engaging the lecture was on a Likert Scale from 1 (not engaging) to 10 (very engaging). This data, along with the gaze data collected from the eye tracker, allowed the researchers to create a machine learning model that could predict how engaged students will be during the presentation of the material.

Tobii Glasses were used by Rosengrant et al. to track student eye movements during a presentation in order to identify causes for inattention [9]. During a physical science lecture, they used eight participants and recorded the locations of where the students were looking. Their findings included that students rarely paid attention to the professor unless he or she was expressing emotion, drawing something, being amusing, or making comparisons that aren't on the presentation slides. New slides tended to maintain or divert student attention to the board.

They preferred to look at drawings or diagrams before reading text. Students who had printed notes before class tended to pay less attention in class, according to the researchers.

With the intent of forecasting the "interest level" and "perception of difficulty," Zhu et al. used sensors on wearable computers to capture both hand gestures and heart activity [10]. During two lectures with 14 topic periods, they requested 30 volunteers to wear Moto 360 timepieces. They gathered motion data at 25 Hz and PPG data at 12.5 Hz, as well as survey data from students with regard to their degree of interest and perception of difficulty in each lecture topic. The researchers found that employing wrist-worn smartwatches for attention monitoring delivers high accuracy, and that using other physiological sensors could potentially be used to improve the accuracy.

Tabassum et al. used a deep learning convolution neural network and the outputs from a cloud-based emotion detection service (Amazon Rekognition) [2]. They gathered data by recording students' webcam photos during a class lecture. The labeled facial photos were used to train a convolutional neural network. Amazon's Rekognition technology was then used to extract the photos' facial emotions. The facial emotions were shown to be statistically significant, indicating that facial emotions can be used to improve the accuracy of attention detection models.

Sanders et al. NiCATS [5] include support for “comprehension analysis” not present in traditional eye tracking software. The tool provided support for gaze metric analysis (e.g., fixation, saccades, heatmap overlays) in the context of both comprehension and the perceived attentiveness of students using images captured via a webcam.

III. Proposed Approach

Fig. 3 provides a high level overview and flow of the NiCATS system with respect to collecting and processing data for knowledge comprehension. The Data Collection illustrated in Fig. 3 primarily resides on the student computer. The hardware setup (Fig. 1) shows the physical placement of the data collection devices used to collect data (eye tracker at the bottom of the machine and webcam at the top). The lightweight application collects screenshots at preset intervals or earlier if user activities such as mouse clicks or scrolls are detected. This is to ensure that screen content and subsequent changes to the screen content are accurately captured. The stream of raw gaze points are also captured using the lightweight application. Both raw gaze points and screen capture are time stamped so they can be synchronized for the evaluation steps. The collected data is then forwarded and stored in a database on the server.

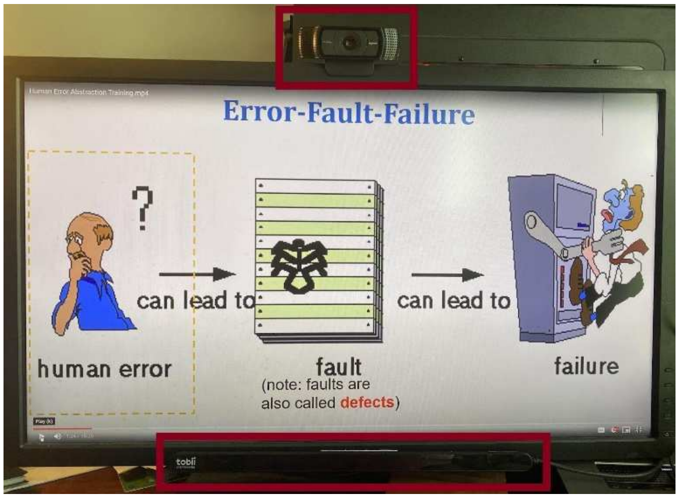
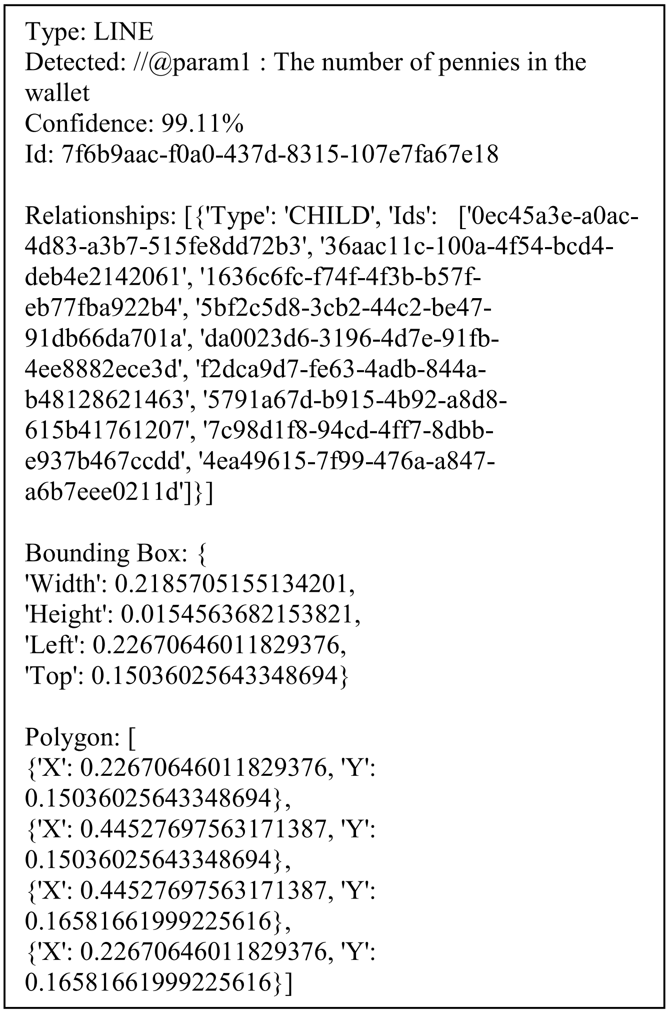

Human Error Abstraction Training.mp4
Error-Fault-Failure
?
can lead to
can lead to
human error
fault
(note: faults are also called defects)
failure
tobii

Fig. 1. NiCATS Data Collection Hardware Setup

The *Pre-Processing* block resides on the server. The student screenshots data are pulled from the server and piped to AWS Textract, a cloud service that automatically extracts text from images, in this case our screenshots, along with coordinate boundaries for each text object identified in the image. The output of the text extraction (Fig. 2) is used to generate ROI maps. There are two types of ROI maps generated, *Lexer* Based (used interchangeably in this paper with Textracts’ *Word* type) which are the regions defined on a word-by-word basis, and *Line* Based which are the regions defined as an entire line of textual content. It is important to note that a *Lexer* based region is not always a word but can often by a single character in the context of code review.



```
Type: LINE
Detected: //@param1 : The number of pennies in the
wallet
Confidence: 99.11%
Id: 7f6b9aac-f0a0-437d-8315-107e7fa67e18

Relationships: [{'Type': 'CHILD', 'Ids':   ['0ec45a3e-a0ac-
4d83-a3b7-515fe8dd72b3', '36aac11c-100a-4f54-bcd4-
deb4e2142061', '1636c6fc-f74f-4f3b-b57f-
eb77fba922b4', '5bf2c5d8-3cb2-44c2-be47-
91db66da701a', 'da0023d6-3196-4d7e-91fb-
4ee8882ece3d', 'f2dca9d7-fe63-4adb-844a-
b48128621463', '5791a67d-b915-4b92-a8d8-
615b41761207', '7c98d1f8-94cd-4ff7-8dbb-
e937b467ccdd', '4ea49615-7f99-476a-a847-
a6b7eee0211d']}]

Bounding Box: {
'Width': 0.2185705155134201,
'Height': 0.0154563682153821,
'Left': 0.22670646011829376,
'Top': 0.15036025643348694}

Polygon: [
{'X': 0.22670646011829376, 'Y':
0.15036025643348694},
{'X': 0.44527697563171387, 'Y':
0.15036025643348694},
{'X': 0.44527697563171387, 'Y':
0.16581661999225616},
{'X': 0.22670646011829376, 'Y':
0.16581661999225616}]
```

Fig. 2. NiCATS Data Collection Hardware Setup

Separately, the raw eye gaze points are used to compute eye metrics data such as saccades and fixations. Each extracted saccade consisted of an (x, y) coordinate for both the start and end points of the saccadic movement along with the duration and timestamp for which the saccade occurred. Similarly, each fixation point consists of a duration and a timestamp, but only a single (x, y) coordinate is stored.

The ROI maps, saccades and fixations elements are used as input for computing Hit/Miss Filtering, which is used to eliminate off-target eye metrics. From the hit/miss data, we compute the Coordinate to ROI translations, which gives the ability to extract comprehension related eye metrics from the perspective of each ROI. These metrics are then used to generate various outputs for instructors including Transition Frequency tables, which are used to assess the most commonly occurring fixation transitions between ROIs, and Region Difficulty Maps, which use the fixation durations order to provide instructors with a visual representation of the most difficult regions for the students to comprehend.

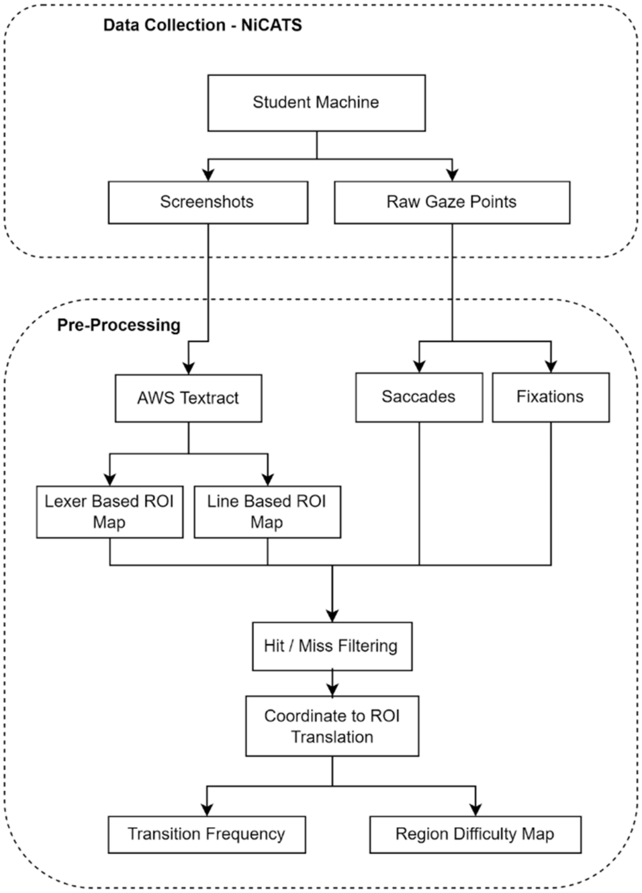


Fig. 3. Overview of NiCATS data collection and data processing

IV. Evaluating Approach

This section provides an overview of the experiment design, including the research goal, experiment setting, artifacts, experiment setup, data collection, and data processing.

A. Research Goal

The goal of this experiment was to explore how instructors can utilize NiCATS data (gaze metrics, screenshots) to gain insights into students' cognitive processes when doing code review tasks.

B. Experiment Setting

Our research took place in the second session of a sequence of volunteer undergraduate computer science students taking an introductory programming course and 10 students participated in the study. The NiCATS software and the associated data capturing gear were installed on each participant's PC to set up the experiment. During the trial, each participant was given four different Java programs to review. Participants were then asked to identify which line numbers contained an error, as well as their explanation for the error.

C. Artifacts

Participants were asked to review four separate Java programs, each with varying complexity metrics, number of lines, and number of seeded defects. Java was selected as the programming language as it is the most familiar language to the participants. Each participant had between six months to one year of experience writing programs in Java. The program examples included bugs of varying difficulty and related to the course content previously taught to the participants. For example, basic syntax errors can often be spotted by reviewing only a single line of code and are relatively easy to identify even for a novice programmer while errors related to object inheritance and data structures often require the participant to review multiple lines of code and are more difficult for novice programmers to identify.

D. Experiment Setup

The following section describes the experiment setup procedure.

Step 1 - Calibration: After logging in to their machine, all participants were asked to calibrate the mounted eye tracker using the Tobii Eye Tracking Core Software which allows the participant to save a calibration profile specific to their own eyes. This improves the accuracy and precision of the collected gaze points and is an essential step for calculating eye metrics against regions of interest with tighter boundaries.

Step 2 - NiCATS Client: The participants were instructed to download the NiCATS client software to their machine. Participants were then asked to launch the NiCATS software which opens a pop-up window describing the personal data that will be collected during the experiment as well as the option to “opt-in” or “opt-out” of the experiment.

Step 3 - Initialize Data Collection: To begin the data collection process, the researchers started a new recording session using the NiCATS web client. This action automatically notifies all machines in the room to start collecting data if the participant is opted-in for the study. Throughout the recording session, raw data points (gaze-point coordinates, screenshots) are collected and stored on the server for post-processing.

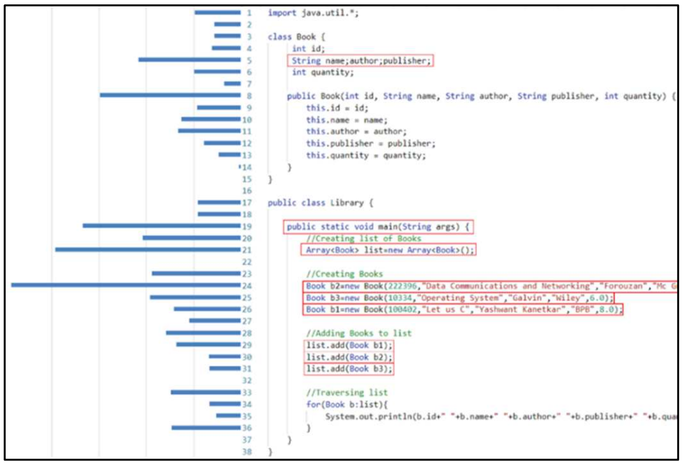
Step 4 - Program Presentation: The participants were given five minutes to review each Java program. Each participant reviewed the programs in full screen mode. This is essential to the experiment because it gives the ability to define ROI layouts to a singular screenshot and analyze every participant’s gaze point data against the same map.

Step 5 - Gathering Responses: After each of the five-minute intervals, the participants were asked to write down which line numbers contained an error, as well as their justification for the error. This step was repeated for all four code examples.

Step 6 - Ending Data Collection: After the four code examples were reviewed, the students were asked to close the NiCATS software, and the researchers ended the recording session via the NiCATS web client.

E. Data Processing

The gaze data and screenshots collected during the experiment have little meaning in their raw form (e.g., fixation durations on a source code sample with highlighted ROIs that contained errors in Fig. 4). Instructors will need to speculate post-hoc based on the data collected from the screenshots and raw gaze points. This highlights the current limitations and the motivation for multilayered and automated analysis presented in the remainder of the paper.

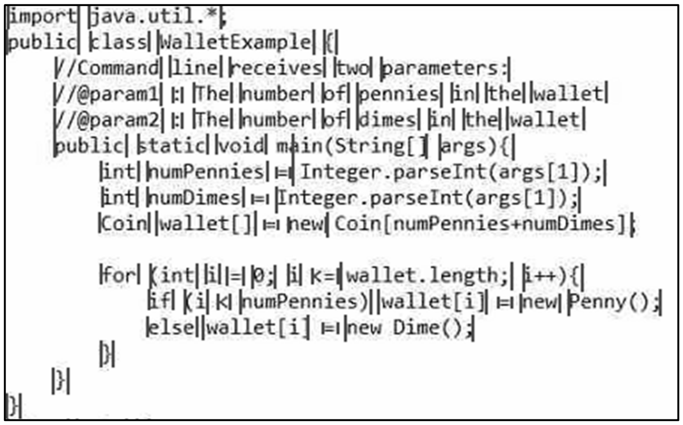


```
import java.util.*;

class Book {
    int id;
    String name;author;publisher;
    int quantity;

    public Book(int id, String name, String author, String publisher, int quantity) {
        this.id = id;
        this.name = name;
        this.author = author;
        this.publisher = publisher;
        this.quantity = quantity;
    }
}

public class Library {

    public static void main(String args) {
        //Creating list of Books
        Array<Book> list=new Array<Book>();

        //Creating Books
        Book b2=new Book(222396,"Data Communications and Networking","Forouzan","Mc G
        Book b3=new Book(10334,"Operating System","Galvin","Wiley",6.0);
        Book b1=new Book(100402,"Let us C","Yashwant Kanetkar","BPB",8.0);

        //Adding Books to list
        list.add(Book b1);
        list.add(Book b2);
        list.add(Book b3);

        //Traversing list
        for(Book b:list){
            System.out.println(b.id+" "+b.name+" "+b.author+" "+b.publisher+" "+b.qua
        }
    }
}
```

Fig. 4.   Fixation count distribution for manually defined ROIs

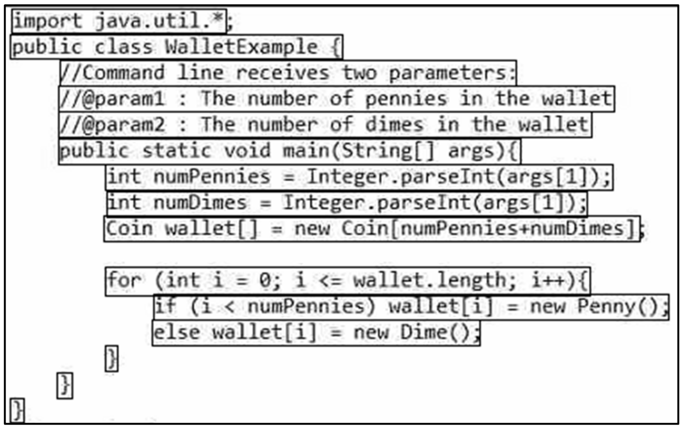
Defining ROI Boundaries: Since each participant reviewed the programs in full screen mode using monitors with the same resolution, we can select a singular screenshot for each question and assign region of interest boundaries to the image. In many code comprehension studies, the regions are defined using McCabe’s cyclomatic complexity metric which measures the number of linearly independent paths through a program module [6]. While useful, Regions defined in this manner often contain many lines of code and many unique lexical tokens which can lead to lackluster results, especially for instructors of introductory programming courses. By defining regions using a multi-line methodology, as seen in complexity-based definitions, instructors are unable to extract the more granular insights associated with these region boundaries. That is to say, comprehension insights extracted from these regions will not include the behavior exhibited within the region boundaries as well as the fixations that occur outside of these boundaries.

To automatically define more granular regions of interest, the screenshots collected by NiCATS are first uploaded as an Amazon Web Services (AWS) S3 object. The S3 objects are then processed using the AWS Textract API. The Textract API is an Optical Character Recognition (OCR) service that accepts an image and returns the *Line* and *Word* boundaries detected of text contained within the image as well as the textual content contained within the region.



```
import java.util.*;
public class WalletExample {
    //Command line receives two parameters:
    //@param1 : The number of pennies in the wallet
    //@param2 : The number of dimes in the wallet
    public static void main(String[] args){
        int numPennies = Integer.parseInt(args[1]);
        int numDimes = Integer.parseInt(args[1]);
        Coin wallet[] = new Coin[numPennies+numDimes];

        for (int i = 0; i <= wallet.length; i++){
            if (i < numPennies) wallet[i] = new Penny();
            else wallet[i] = new Dime();
        }
    }
}
```

Fig. 5. Word based region of interest map (vertical lines indicate start and end of word)



```
import java.util.*;
public class WalletExample {
    //Command line receives two parameters:
    //@param1 : The number of pennies in the wallet
    //@param2 : The number of dimes in the wallet
    public static void main(String[] args){
        int numPennies = Integer.parseInt(args[1]);
        int numDimes = Integer.parseInt(args[1]);
        Coin wallet[] = new Coin[numPennies+numDimes];

        for (int i = 0; i <= wallet.length; i++){
            if (i < numPennies) wallet[i] = new Penny();
            else wallet[i] = new Dime();
        }
    }
}
```

Fig. 6. Line based region of interest map (each line bounded by a box)

Gaze Data to Fixations: Once the ROI maps are defined for each question, we process the raw gaze data in the context of the three ROI maps to extract the relevant eye metrics.

1) *Fixations:* Fixations were calculated from the raw gaze data using the I-DT algorithm [20] with a 1° maximum dispersion and 300+ millisecond duration threshold. After the region boundaries are defined and fixation coordinates are extracted, the coordinate is boundary checked against each region of every region of interest map to check if the fixation is a *hit* or a *miss*.

a) A *hit* refers to a fixation (shown as a solid circle) that occurred inside the boundaries, or within 1° (to account for accuracy error in the eye tracking hardware) of any region of interest in our predefined map (Fig. 7 and Fig. 8).

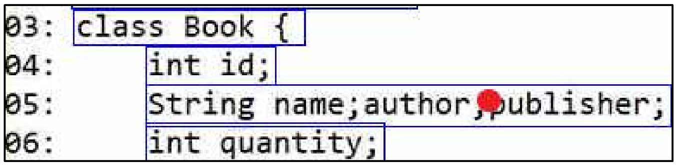
b) A *miss* refers to any fixation that lies outside the boundaries of all regions in our region of interest map (Fig. 9). Any fixation identified as a *hit* is stored in the database and any fixation identified as a *miss* is omitted. The *miss* fixations are omitted because these regions contain no text which means, in this experiment setting, that this fixation is not aiding the student with their comprehension.

2) *Additional Metrics:* Using the results of the hit/miss boundary detection, we calculate several essential metrics to be used in our analysis. On a participant to question level, we calculate the following with respect to each ROI.

a) Average Fixation Duration: The total fixation time for an ROI divided by the number of fixations in that ROI.

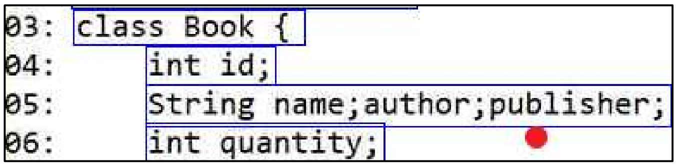
b) Number of Fixations: The total number of times a participant fixated in an ROI. This is a common metric used in eye tracking studies as an indicator of how much visual attention is required to comprehend the information [4].

c) ROI transition frequency: Calculated as the most frequently occurring fixation transition paths between ROIs between the entire sample population. Fixation transitions provide insight into the *links* made by the participants between regions of interest. This provides insight into the ordering for which ROIs were used by the students to comprehend portions of the code base.



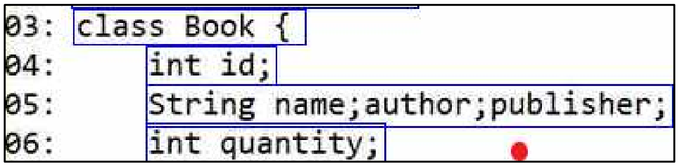
```
class Book {
    int id;
    String name;author;publisher;
    int quantity;
```

Fig. 7. Fixation hit (Direct)



```
class Book {
    int id;
    String name;author;publisher;
    int quantity;
```

Fig. 8. Fixation hit (1° offset)



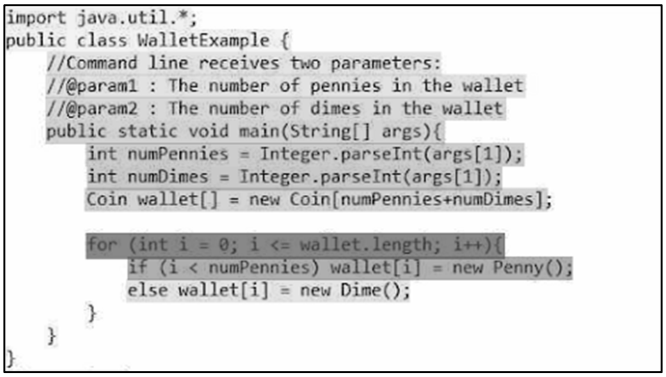
```
class Book {
    int id;
    String name;author;publisher;
    int quantity;
```

Fig. 9. Fixation Miss

V. Results and Discussion

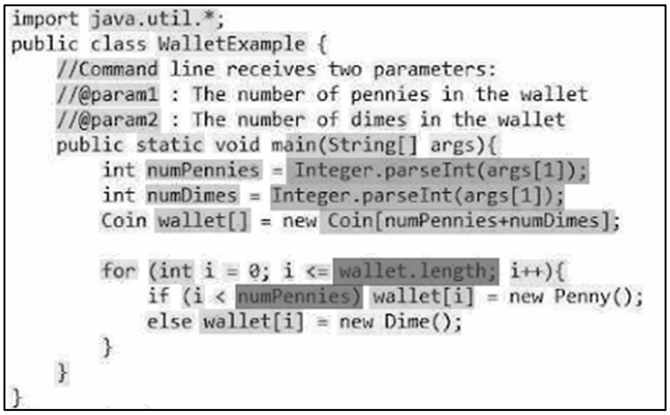
We performed qualitative analyses of the results in the context of individual students and across the entire population in order to extract insights related to the comprehension processes of the students.

Using the extracted fixation durations for each region of interest (both the *Line* based map and the *Word* based map), instructors can use the generated fixation duration overlay to gain insight into the comprehension difficulty of each region of interest. Fig. 10 and Fig. 11 illustrate these fixation durations for all participants on a cropped code sample used in the experiment. Instructors reviewing the *Line* based graphic can quickly identify that students spent significantly more time fixating on the lines contained within the *for* loop (indicated by a darker highlighted region). Further review of these fixation durations using the *Word* based ROI map indicates which specific words in these *Line* based regions were most frequently fixated.



```
import java.util.*;
public class WalletExample {
    //Command line receives two parameters:
    //@param1 : The number of pennies in the wallet
    //@param2 : The number of dimes in the wallet
    public static void main(String[] args){
        int numPennies = Integer.parseInt(args[1]);
        int numDimes = Integer.parseInt(args[1]);
        Coin wallet[] = new Coin[numPennies+numDimes];

        for (int i = 0; i <= wallet.length; i++){
            if (i < numPennies) wallet[i] = new Penny();
            else wallet[i] = new Dime();
        }
    }
}
```

Fig. 10. Fixation Duration density on *Line* based ROI map



```
import java.util.*;
public class WalletExample {
    //Command line receives two parameters:
    //@param1 : The number of pennies in the wallet
    //@param2 : The number of dimes in the wallet
    public static void main(String[] args){
        int numPennies = Integer.parseInt(args[1]);
        int numDimes = Integer.parseInt(args[1]);
        Coin wallet[] = new Coin[numPennies+numDimes];

        for (int i = 0; i <= wallet.length; i++){
            if (i < numPennies) wallet[i] = new Penny();
            else wallet[i] = new Dime();
        }
    }
}
```

Fig. 11. Fixation Duration density on Word based ROI map

In the previous example, due to the experience level of the students as well as the nature of looping concepts in programming, the results for the distribution of these fixation durations were expected. That is to say, for both the *Line* and *Word* based ROI maps, the researchers expected to find longer fixation durations in the regions contained within the *for* loop.

In addition to the fixation duration distributions, instructors also have the ability to view the transition frequencies of dynamic length between ROIs on an individual and classroom level. By reviewing these transitions, instructors can identify which regions students are using as well as the ordering of these regions, to evaluate *how* students are comprehending the material.

In the context of the above example, it was found that the most frequently occurring fixation transition sequence for all participants also occurred within this *for* loop. The output generated by the program using the *Line* based ROI map with n=3 is shown in Table 1, whereas Table 2 shows the output generated using the *Word* based ROI map with n=3. Reviewing the results of these ROI transitions, the instructor can see that, on a *Line* level, students frequently transitioned between the *for* loop declaration statement, and the following line. Instructors can then make data driven insights that students were following the iterative procedure of the *for* loop to determine if any errors existed in these regions. Additionally, a review of the transitions at the *Word* level indicate that the students most frequently transitioned between *wallet.length* (an upper boundary control for limiting the number of loop iterations) and *i++* (the incrementing value for the loop). This is an indicator that students were frequently checking the control parameters of the *for* loop as they attempted to identify any existing errors in the loop.

TABLE I. MOST FREQUENTLY OCCURRING ROI TRANSITION BETWEEN ALL STUDENTS (LINE BASED: N=3)

ROI Text	Frequency
1. for (int i = 0; i <= wallet.length; i++) 2. if (i < numPennies) wallet[i] = new Penny(); 3. for (int i = 0; i <= wallet.length; i++)	35

TABLE II. MOST FREQUENTLY OCCURRING ROI TRANSITION BETWEEN ALL STUDENTS (WORD BASED: N=3)

ROI Text	Frequency
1. wallet.length; 2. i++) 3. wallet.length;	35

The assessment of these results using only the previously discussed outputs is an indicator that the students, in general, were all comprehending the *for* loop correctly. By following the *for* loop iteratively while checking boundary conditions for the loop itself, students indeed understand the looping concepts well enough to identify certain runtime errors that may exist. To validate this, the quiz responses containing the line number and explanation for each identified error were evaluated to see if any student reported an error in the four lines used in the *for* loop. It was found that no student reported an error in any of these regions.

VI. LIMITATIONS AND SUMMARY OF RESULTS

The results of this approach do have some limitations that must be addressed. While using the AWS Textract API to automatically define regions of interest does eliminate a significant amount of the manual work required in previous NiCATS experiments, Textract does not always accurately identify the regions contained in the image. For example, in Fig. 11 we can see that the region boundaries for *main(String[]* are cut off where the leftmost boundary of this word divides the letter *m* in half. The result of this accuracy error outputs the word *nain(String[]* instead of the true text that should have been included. Another example not shown in these figures consists of an error where the *Line* based detection does not always include all of the necessary text to define the region in its entirety. This was a common occurrence for regions where 2 or more spaces were present between words.

Another limitation of this study was the experience level of the participant. The average score for the test results across the student population was 35.94% which indicates that many of the students struggled with identifying errors in the code.

These limitations could have a significant impact on the interpretation of the results if the data is analyzed without the context of the screenshot. However, for this study, we are able to review the screenshot to visually identify where these errors exist and the errors can be accounted for when reviewing the results.

To summarize the results, it can be said that the method used for defining regions of interest has a significant impact on the way the results can be interpreted by an instructor. The findings in this research show that using multi-layered ROI maps can provide instructors with unique insights into the cognitive processes of students that are heavily influenced by the methodologies used for defining the ROI map initially. Using a *Line* based definition, we were able to see that students had more difficulty comprehending the *for* loop than any other regions in the code (as seen by the *Line* based fixation duration) which was further supported by the most frequently occurring fixation transitions for this ROI map. Using the *Word* based definition, it was found that students had most difficulty comprehending two of the words bounded by these *Line* based regions (one in each row), however, when reviewing the transition frequency for this code sample, the results indicate that the most frequently occurring transition was not between the two lines of code, but between two *Words* located within the same line of code. To summarize these results, it can be concluded that using various ROI definitions provides essential information needed to identify and assess the comprehension patterns of students doing code review tasks.

VII. Conclusion and Future Work

This paper presents the investigation, design, analysis, and results of an exploratory work for the developing an automated pedagogical toolset for assessing student comprehension while performing code review tasks. This methodology is an expansion to our existing NiCATS application through the addition of a comprehension analysis toolset which can be used by instructors to evaluate student performance.

The novel contribution of this paper is the expansion of the NiCATS data collection system to accommodate instructors by providing pedagogical tools for use in comprehension assessments. Another novel contribution is through the use of multi-layered ROI mapping mechanisms to provide user-defined, granular insights about student comprehension patterns that are not presently available in state of the art educational tools. This research could be invaluable to instructors seeking a quick solution for identifying key points of misunderstanding among their students. By identifying these problem areas early on, instructors can use this information to design educational interventions in order to improve student knowledge acquisition and knowledge retention.

References

[1] J. Whitehill, Z. Serpell, Y. Lin, A. Foster and J. R. Movellan, "The Faces of Engagement: Automatic Recognition of Student Engagementfrom Facial Expressions," in IEEE Transactions on Affective Computing, vol. 5, no. 1, pp. 86-98, 1 Jan.-March 2014, doi: 10.1109/TAFFC.2014.2316163.

[2] Tabassum, T., Allen, A. A., & De, P. (2020). “Non-Intrusive Identification of Student Attentiveness and Finding Their Correlation with Detectable Facial Emotions.” Proceedings of the 2020 ACM Southeast Conference, 127–134. Presented at the Tampa, FL, USA. doi:10.1145/3374135.3385263

[3] Veliyath, N., De, P., Allen, A. A., Hodges, C. B., & Mitra, A. (2019). “Modeling students’ attention in the classroom using eyetrackers.” Proceedings of the 2019 ACM Southeast Conference, 2–9. doi:10.1145/3299815.3314424

[4] Sharafi, Z., Shaffer, T., Sharif, B., & Guéhéneuc, Y.-G. (2015). “Eye-tracking metrics in software engineering.” 2015 Asia-Pacific Software Engineering Conference (APSEC), 96–103. doi:10.1109/APSEC.2015.53

[5] A. Sanders, B. Boswell, G. S. Walia and A. Allen, “Non-Intrusive Classroom Attention Tracking System (NiCATS),” 2021 IEEE Frontiers in Education Conference (FIE), 2021, pp. 1-9, doi: 10.1109/FIE49875.2021.9637411.

[6] H. Hijazi, J. Cruz, J. Castelhano, R. Couceiro, M. Castelo-Branco, P. d. Carvalho and H. Madeira, “iReview: An Intelligent Code Review Evaluation Tool using Biofeedback,” in the 32nd International Symposium on Software Reliability Engineering (ISSRE 2021), 2021

[7] P. Rodeghero and C. McMillan, “An Empirical Study on the Patterns of Eye Movement during Summarization Tasks,” 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 2015, pp. 1-10, doi: 10.1109/ESEM.2015.7321188

[8] Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. ”Using psycho-physiological measures to assess task difficulty in software development.” In Proceedings of the 36th International Conference on Software Engineering(ICSE 2014). Association for Computing Machinery, New York, NY, USA, 402–413. doi: 10.1145/2568225.2568266

[9] Rosengrant, D., Hearrington, D., Alvarado, K., & Keeble, D. (2012). “Following student gaze patterns in physical science lectures.” AIP Conference Proceedings, 1413(1), 323–326. doi: 10.1063/1.3680060

[10] Zhu, Z., Ober, S., & Jafari, R. (2017). “Modeling and detecting student attention and interest level using wearable computers.” 2017 IEEE 14th International Conference on Wearable and Implantable Body Sensor Networks (BSN), 13–18. doi: 10.1109/BSN.2017.7935996

[11] J. H. Goldberg and X. P. Kotval, “Computer interface evaluation using eye movements: methods and constructs,” International Journal of Industrial Ergonomics, vol. 24, no. 6, pp. 631–645, 1999.