Array 15 (2022) 100241

Contents lists available at ScienceDirect

Array

journal homepage: www.sciencedirect.com/journal/array

Eye tracking technologies to visualize secure coding behavior

Daniel Kyle Davis *, Feng Zhu

University of Alabama in Huntsville, Computer Science Department, 301 Sparkman Dr NW, Huntsville, AL 35899, United States

ARTICLE INFO

Keywords:
Cybersecurity secure coding
Hands-on education learning
Software development coding behaviors
Eye tracking
Eye tracking user-interactive stimuli
Software vulnerabilities

ABSTRACT

Secure coders' experiences and performances vary greatly and any missed security flaws in source code may lead to costly consequences. Their behavior to analyze source code and develop mitigation techniques is not well understood. Our objective is to gain insight into the strategies and techniques from both novice and experienced developers. Proper understanding can help us to inform inexperienced coders to efficiently and accurately approach, discover, and mitigate security flaws. Our research relies upon eye tracking hardware and software to collect and analyze the eye gazes. Unlike existing approaches, we incorporate a wide range of tasks simultaneously reading documentation, writing code, and using security coding analysis tools. We analyze both static and dynamic (interactive) stimuli in a realistic software development environment. Our pictorial visualizations represent a coder's eye gazes that visually demonstrates their behavior and patterns. In addition, we provide the full context of the stimuli that a participant observed. This allows for investigating the behavior at a range of tasks for a single participant and between participants. Our secure coding tasks include reading documentation, reading source code, and writing source code for a web application as well as utilizing security code scanning tools. Our contributions also include (1) novel visualization techniques to present transitions among components within and between applications, and (2) presentations of coders' attention levels during secure coding by investigating the change of pupil sizes. The eye tracking collection and analysis techniques support both modifiable stimuli and stimuli presented in different sequences based upon individual participant's behavior.

1. Introduction

Secure coders often examine various aspects of a programing code: syntax, semantics, program flow, and overall program execution. They typically focus on finding security weaknesses and then determine the appropriate mitigation techniques. With diverse skills, secure coders may approach discovering and fixing vulnerabilities very differently and inefficiently. While secure coding becomes increasingly important, secure coding behavior is not well understood. Understanding secure coding behavior and where challenges occur during coding can help us understand where to improve learning content and therefore improve secure coders abilities.

Existing methods of investigating behaviors often utilize think-aloud sessions, questionnaires, or verbal reviews, but secure coders ability is stored in skill memories, which cannot always be verbalized [1]. Therefore, it is difficult to collect and analyze behavior and patterns via questionnaires or verbal reviews in a non-bias or objective manner. Direct observations do not require participants to recall or verbally explain their techniques during secure coding, but it is unnatural and distracting to participants [2]. A discreet and unobtrusive technique is needed to observe secure coders behaviors while not requiring participants to recall or verbally explain their techniques during secure coding.

Eye tracking technologies allow researchers to observe participants' eye gazes objectively. The analysis of eye tracking results allows for investigating participants' reading patterns and behavior across distinct types of visual stimuli without requiring participants' to verbally communicate their actions. Most existing eye tracking technologies focus on top-down and left-to-right reading pattern but reading source code and performing secure coding activities does not follow the standard text reading patterns or even general software coding patterns. Existing visualization methods for eye gaze analysis have limitations for eye tracking stimuli that is user driven and when multiple types of eye tracking stimuli are presented at a same time.

The fundamental challenge stems from user-controlled eye tracking stimuli. Secure coders may read paragraphs of documentation and guidelines; they may read and write source code; and they may utilize security scanning tools. Often secure coders perform several of these tasks concurrently with an application visually adjacent to other applications and rapidly switch focus between applications. Furthermore, the

* Corresponding author.
E-mail addresses: dkd0004@uah.edu (D.K. Davis), fz0001@uah.edu (F. Zhu).

https://doi.org/10.1016/j.array.2022.100241
Received 5 April 2022; Received in revised form 27 July 2022; Accepted 28 July 2022
Available online 18 August 2022

Abbreviations

AOIs	Areas of Interest
CERIAS	Center for Education and Research in Information Assurance and Security
CAPEC	Common Attack Pattern Enumeration and Classification
CWE	Common Weakness Enumeration
ETRA	Eye Tracking Research and Applications Symposium
NVD	National Vulnerability Database
OWASP	Open Web Application Security Project
RIPS	Research and Innovation to Promote Security
SVG	Scalable Vector Graphics
SMA	Simple Moving Average
SEI	Software Engineering Institute

process and order of finding and mitigating security weaknesses may vary for each participant.

Our research goal is to understand secure coding behaviors and overcome the limitations of existing eye tracking technologies. That is, we enable (1) complex tasks and long sessions that are driven by user tasks, (2) multiple types of stimuli/applications simultaneously, (3) visualization and comparison of multiple participants. In this paper, we also compare the existing techniques and propose our novel eye gaze visualization approaches to observe participants' secure coding behavior. Visualizing and understanding the behavior of secure coders can allow us to discover barriers in our hands-on learning modules and improve the learning content for educating secure coders.

Our study provides a realistic software development environment. Participants can scroll in documents and source code files, switch among multiple source code files, use a static analysis tool, and conduct web searches. Our design allows each participant to control the flow of fixing vulnerabilities. This dynamic design setup is essential for the objective of providing a natural software coding environment that is needed for evaluating the value of eye gaze visualization techniques for secure coders and their coding behavior.

Our main contribution is the understanding of secure coders' behaviors with multiple types of visualizations of distinct aspects in secure coding and over a timeline. Our approach considers the linear interval of time and sequence of patterns that secure coders transition between stimuli and remain within stimuli. Our visualization techniques, especially those that present different stimuli content concurrently, are essential to understand the secure coding behavior. At the low level, we process eye movements, the speed of movement, the duration of eye fixation, and changes in pupil sizes. At the medium level, we examine participants' eye gaze at the applications and source code files or function level. At the high level, we present participants' secure coding patterns and strategies.

Besides the adaptation of existing eye tracking technologies for secure coding, we propose swimlane diagrams, state transition diagrams, and pupil size fluctuation diagrams. These allow us to investigate the transitions within multiple components of source code files and other applications. The swimlane diagrams and state transition diagrams provides the ability to group common software elements together and therefore visually determine and objectively quantify when secure coders have significant changes to different groupings of AOIs over a timeframe. In addition, we capture coders' attention (the changes in eye gaze) over a timeline to understand and compare their efforts.

The rest of this paper is organized as follows. Section 2 introduces general information about secure coding and educational material concerning secure coding, general information about eye tracking, and a summary of existing visualization approaches and their limitations. Section 3 reviews the type of stimuli presented in our hands-on secure coding learning modules. Section 4 contains information about our eye tracking study and data collection. Section 5 discusses the data preparation that is required to transform raw eye gaze data points into fixations and the creation of Areas of Interest (AOIs). Then, Section 6, contains several of the visualization diagrams based upon the type of stimuli utilized in our secure coding exercises. Section 7 discusses the validation procedure that was followed to examine how well participants responded to the various questions or programming problems. Finally, Section 8 concludes with a summary and discussion on our findings of existing visualizations and provide an overview on some of the limitations of existing approaches.

2. Related work

2.1. Secure coding resources

Software vulnerabilities can exist in several aspects of source code: the lack of focus on security threats when an application is developed, using open source or third-party components, being more interconnected, and sharing user data over network devices and accounts [3]. Security vulnerabilities damage integrity, confidentiality, and availability. Secure coding is a major aspect to software assurance and vulnerability management [4,5]. It is better to develop source code that is secure during the software development lifecycle before publication as the cost to mitigate after software is published is typically higher. Resources that help with secure coding includes vulnerability databases [6–8], code reviews [9,10], secure coding standards [11–14] and as well as tool suites [15–17]. However, there is a lack in understanding a secure coder behavior when the coder applies the various techniques or utilizes the secure coding resources. These methods could potentially be improved by an understanding of secure coding behavior and how coders approach secure coding problems [18]. Our research is focused on systematically studying the behavior of secure coders in order to bring improvements to the secure coding process.

2.2. Secure coding education

The creation of learning content and hands-on learning modules to help facilitate teaching is a fundamental and critical component of learning [19,20]. Various industry researchers and university scholars have contributed towards developing content for learning secure software development principles. The Software Engineering Institute (SEI) at Carnegie Mellon University has created several curriculums based upon the experience level of participants. Each curriculum contains a set of tools, procedures, and recommended tactics for how to develop learning material to help secure coders discover and mitigate security weaknesses in software applications [21]. Researchers at Purdue University have created the Center for Education and Research in Information Assurance and Security (CERIAS) with several principal investigators researching secure programming education [22]. Researchers at Towson university have developed workshops aimed at security injections in source code [23]. Georgia Tech has created the Secure Software Development course that is offered as a three-day online course [24]. At Syracuse University, researchers have developed the SEEDLABS 2.0 to provide tasks in labs and workshops, video lectures, and references for common vulnerabilities in software [25]. In 2021, Open Web Application Security Project (OWASP) released their secure coding training platform identified as Dojo after it was given to OWASP from Trend Micro [26]. SAFECode provides security training courses that are publicly available as webcast [27]. Other companies provide secure coding training; however, several companies typically often have a fee for the complete suite of training and tools [28–31].

Our approach is to educate students with free and publicly available learning modules that guide students step-by-step, but still provide the opportunity to mitigate security coding flaws with hands-on experience. We developed our modules to help coders learn practical strategies by first identifying and then mitigating vulnerabilities in source code. Our

hands-on approach involved using a photo and video web sharing application. We used vulnerabilities from the Common Weakness Enumeration (CWE) repository to facilitate participants learning about weaknesses and vulnerabilities that are determined by industry to have a high importance [8]. The CWE repository provides a list of common weaknesses and vulnerability types as well as example code and exploits that could apply to several programming languages [32]. We also introduced specific CWEs from the OWASP Top 10 list and the National Vulnerability Database (NVD) [33,34]. Our learning modules illustrated coding instances of specific CWEs and showed exploitation methods and security mitigation techniques. Finally, each learning module concluded with an overall discussion. Currently, one of our objectives is to understand how participants use our learning modules when they are working secure coding exercises.

Table 1
Eye tracking visualization methods.

Visualization Name	Description	Benefits	Limitations
Heat Map	Color palette allows for the areas with more fixations to be distinguish from the areas with less fixations.	Quickly determine focus area over a stimuli element or AOI.	Does not work well with dynamic video stimuli. Limitations when used for multiple participants.
Scan Path or Gaze Path	Plot eye movements from one fixation to the next fixation forming a path.	Helps to form a time-based aspect to the graphics of viewed stimuli.	Difficult to examine for multiple participants or with a long session for a single participant.
Clustering	Different clusters are created when fixations are spatial separated based upon an analyst supplied parameter.	Works well to give an overview of several participants.	Does not work very well for dynamic video stimuli.
Fixation Matrix	A table of fixations usually presented as a count or as a percentage per stimuli/AOI.	An overall summary of fixation count or percentage.	Does not indicate the chronological order in eye gaze transitions.
Transition Matrix	A table indicating transitions to other stimuli elements/AOIs usually presented as a count of transitions to and from stimuli elements/AOIs or a percentage of the total.	Provides an overall summary of which stimuli/AOI had higher transitions from other stimuli elements/AOIs.	Does not show the order of transitions between the various stimuli elements/AOIs. Limits when the number of stimuli/AOIs increases.
Radial Transition Graph	A circular graph representation of the visited stimuli/AOI. The transition from one stimuli/AOI to another stimuli/AOI is symbolized by arc lines.	Visualize the transitions to and from stimuli/AOI along with the dwell times per each stimuli/AOI.	As number of stimuli elements/AOIs increases and/or the number of transitions increases, then the graph is difficult to understand.
Scarf Plot	Stimuli elements/AOI are marked on a timeline per each participant and can be transition-based or time-based. A color-code is often used for unique identification.	Useful for analyzing the stimuli/AOI sequences or dwell times in the order that they occur.	Difficult to determine differences in patterns as the number of participants increase. Difficult to understand with different temporal ordering of stimuli/AOI.
Parallel Scan Path	Stimuli elements/AOIs are plotted on the X-axis with time on the Y-axis.	Ability to see the transitions over stimuli/AOI for a small number of participants.	Difficult to analyze as the number of stimuli/AOI increases or considerable number of transitions analyzed.

2.3. Approaches to evaluate learning techniques

Often tests or exams are given that attempt to gauge the content that students comprehend from lectures. Essays or projects are another ordinary form in some fields to gauge student understanding. Interviews can provide more information about student thinking processes and their level of understanding that might be difficult to collect with paper exams or essays [35]. Programming assignments are commonly used in software development coursework. Several of these methods require designers of the material to correctly setup the questions and assignments to determine the level of understanding [36]. These methods are often specific to certain fields of study and depend upon the group of participants. These methods also lack the ability to objectively collect the behavior of participants. Our research is focused on the collection and analysis techniques of participants' behaviors as they learn how to develop secure software.

2.4. Eye tracking

Eye tracking technologies permit researchers to collect, analyze, and visualize the viewing behavior of participants (e.g., read text and browse the Internet). Our eyes are comprised of various structures such as our pupil, retina, fovea, and various other components to help us visualize objects and colors [37]. Studying changes of pupil sizes, gaze location, pattern of eye movements allows researchers to examine participants driving behavior [38,39], to determine attention in advertising media [40], to predict attention in playing chess [41], to study skills of flying and landing airplanes [42–44], and to examine software UML class diagrams [45]. Prior research has found that eye tracking alone cannot directly determine the amount of cognitive load on individuals. However, research is ongoing in using eye tracking in combination with questionnaire and various stimuli to examine behavior and patterns of participants [46]. Our research study uses eye tracking technology to study the behavior of secure coders. In our study, students were presented with questions and secure coding exercises while their eye gazes were recorded. This paper focuses on how to visualize the eye gaze of secure coders.

We investigate secure coders eye fixations and transitions with visualizations. A fixation occurs when a participant eye is relatively fixed at one point and their eye has no movement. When viewed on a computer monitor, then a fixation is recorded as a X-point and a Y-point for a 2-dimensional viewing scene along with a timestamp. This allows for researchers to study the eye gaze of participants and determine when in time and where in the scene the eye gaze occurred. This can allow researchers to observe the viewing pattern of participants which is useful to determine the attention that participants have given to specific parts of a reading or visualization stimuli [47]. Transitions and order of stimuli visited can give insight into the behavior and methods that participants used when reading and watching stimuli. When the eyes are rapidly moving between two fixations, a saccade occurs [48]. The length for each saccade is noticeably short and is in the order of a few milliseconds. Fixations typically last for milliseconds to seconds [46]. Saccades are often utilized to help determine the amount of attention or changes in attention [49]. Raw eye tracking data that is collected during an experiment including fixations and saccades can be very time-consuming to process and complex to understand. A commonly adopted technique includes creating Areas of Interest (AOIs) on presented stimuli. This allows a research analyst to add specific regions of interest to the stimuli that was presented to participants [45]. The transitions between AOIs are important to analyze as well as the fixations that occur inside of an AOI. An AOI is generally created manually

by drawing on top of the stimuli content [49]. This can help to identify patterns and tactics that participants may utilize to approach a problem but could also help to identify anomalies.

2.5. Existing eye tracking data analysis and visualization

An important phase after the data collection is the data analysis. The methodologies applied to transform the raw eye fixations and saccades into meaningful insight is lacking with certain stimuli content [48]. Common techniques are appropriate for reading paragraphs of natural reading text; however, these approaches do not work well for stimuli that is more dynamic media-based in nature such as source code reading or scrolling through web-pages [50]. When the order of stimuli is not the same for all participants in the eye tracking recording sessions due to the unique individual behavior of the participant, then existing approaches of data analysis do not work well [51] because the eye movement patterns are much more complex than simpler stimuli content. Furthermore, it is often necessary to determine the visual scan path that participants take to solve a problem in the chronological order in which participants viewed the stimuli [45]. This can be especially beneficial to determining patterns between participants.

Often the eye tracking metrics are measurements associated with time, sequences, or numerical count of fixations either for a recording segment or specific AOIs. Examples of general eye tracking metrics include the first fixation duration, total fixation duration, fixation count, time to first fixation, visit duration or dwell time [49]. The fixation duration measurement provides a time value that participants fixated on a stimuli element or AOI during a recording. The fixation count metric is a measure of how many fixations occur within a recording or AOI. The time to first fixation is the amount of time that has passed until the first fixation. Our selected eye tracking software provides many of these metrics [48].

Switches in attention between and within stimuli or AOIs can help researchers understand the visual strategies over a period that participants used when inspecting the stimuli presented. Visualization methods are needed to fully understand the behaviors and patterns in the transitions between and within stimuli or AOIs. Several common eye tracking data visualization techniques are Heat Maps, Scan Paths and Clustering.

During our research, we explored published literature to determine existing methods and techniques that eye tracking researchers have explored for analyzing eye tracking data. Researchers have created tools for visualization of eye fixations using heat maps, gaze plots and clustering over geographic map views [52]. Researchers often explore the usage of a fixation matrix [53] or transition matrix to provide an overall summary of participant behaviors [54,55]. Some studies focused on analyzing stimuli that is not bound to the same timeframe for each participant [51]. Scarf plots allow for visualization of the AOIs and the associated transitions displayed over a linear timeline [49,51,54]. Radial transition graphs allow for the transition from one AOI to another AOI to be symbolized by arc lines in a graph [54]. Using the radial transition graphs, researchers have discovered patterns of linear code reading as well as patterns representing the execution of source code [50,56]. Researchers have investigated the usage of viewing multiple participant scan paths next to each other and color-coding specific AOIs that are of particular interest [56]. In Table 1, we summarize the major eye tracking visualization methods that is currently published by researchers.

Several existing and common visualization techniques work well for static stimuli content or with stimuli that has a small number of objects to distinctly observe. Eye tracking data for secure coding often includes interactive content that requires scrolling of multiple web pages and multiple source code files as well as participants switching back and forth between multiple files. User interactive content complicates the data analysis phase; however, we believe that for secure coding activities to be realistic, we must allow each secure coder to control the flow of events as each coder works to discover and mitigate security vulnerabilities. We created a custom analysis program to generate additional eye tracking visualizations method that generate new types of images for visualization of eye gaze.

2.6. Programming language discoveries with eye tracking

Prior research using eye tracking to examine when participants read source code has discovered that initial portions of source code tend to have long dwell durations with the code segments viewed later having shorter dwell durations [45,57]. Another study found that the verification of ideas and thoughts on how source code transforms input data is often determined by software developers jumping among functions in the code [58]. Additionally, researchers have recently completed detailed literature survey reports that contain comprehensive descriptions of using eye tracking technologies for general software development activities development [59–65]. Additionally, conferences such as The Eye Movements in Programming Conference and the Eye Tracking Research and Application Symposium now exist that directly focus on eye tracking related research with software programming [41,47,50,55,56,66–72]. Eye tracking studies have examined and determined that differences exist when developers' read source code than when they read natural language text [73]. Recent publications demonstrate that allowing participants to scroll in source code while their eye gaze is recorded is beneficial. However, their tool is limited to specific software development toolkits such as initially the Eclipse integrated development environment [74,75]. The iTrace tool is continuing to receive development support as it has now been extended to work with the Google Chrome web browser [67,76]. Current published literature indicates that additional work is needed for the iTrace tool to support source code that a participant can modify [67,76–78]. Reading and writing source code for general algorithm comprehension is not directly the same as working to discover and resolve security vulnerabilities. However, there are similarities with how developers approach software coding and secure coding [79].

Our focus, in this paper, is researching visualization methods for multiple types of stimuli content including static eye tracking stimuli as well as non-linear dynamic stimuli. This can allow a software developer not only to scroll in long source code files but also to modify the source code that is presented as stimuli elements. Furthermore, we do not want to limit the design to only a specific software development integrated development environment. This provides a more natural environment for a software developer than limiting the stimuli to only lines of code that fit on a single monitor [63]. The current literature is lacking with visualization and analysis methods for activities involved in secure coding particularly for non-linear dynamic stimuli. Many existing eye tracking publications limit participants ability to scroll in the eye tracking stimuli which also includes limiting the ability of a participant to actually modify the eye tracking stimuli that is presented [56,68,79,80]. Our primary focus is directly observing secure coding and learning activities, and then we can determine the potential to analyze secure coder patterns and behavior.

Writing source code and performing secure coding tasks should not mandate a specific step-by-step order of stimuli element content that eye tracking participants must follow, if the desire is to capture the unique self-motivated behavior that each secure coder uses to find and mitigate security flaws. Thus, a need exists to improve the methodologies used to investigate and analyze eye tracking data for software development and secure coding. A potential pitfall with some methods is that it can allow for the unique individual differences in secure coders to be obscured from view. For example, this can occur if the data is simplified as an arithmetic mean for several participants over an extended timeframe [47]. Therefore, the visualization approach may depend upon the type of stimuli content and the task given to coders.

Software development includes many aspects for designing, developing, and maintaining software applications. The areas of software

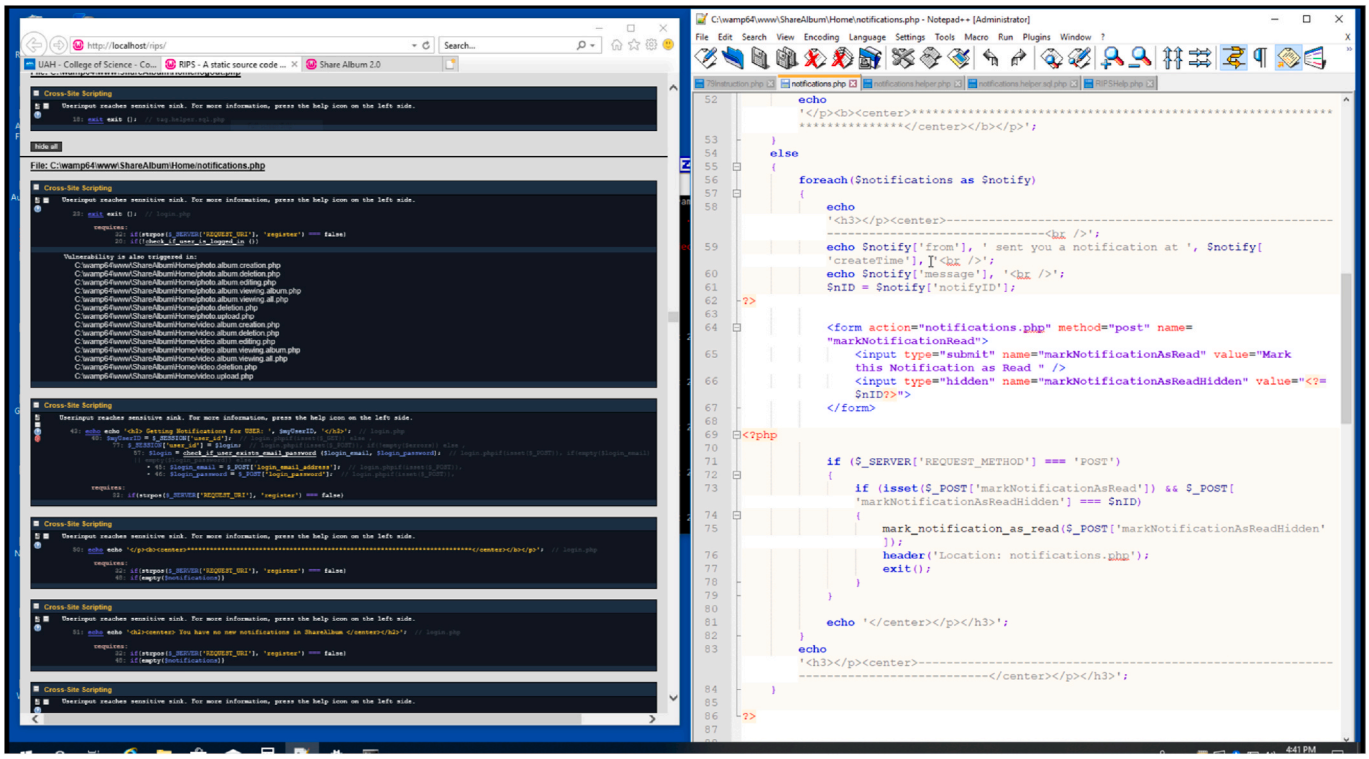


Fig. 1. Split screen between RIPS tool in web browser and code editor.

requirements gathering and documentation [81], software traceability [82–84], graphical modeling languages (UML) [85–88], user interface designs, database interfaces, and software unit testing is beyond the scope of this manuscript. While this paper focuses on the secure software development area of software development, it will be interesting to explore other aspects of the software development domain in the future. Our manuscript includes references to the manuscripts of multiple other researchers' discoveries in these other software development areas for interested readers.

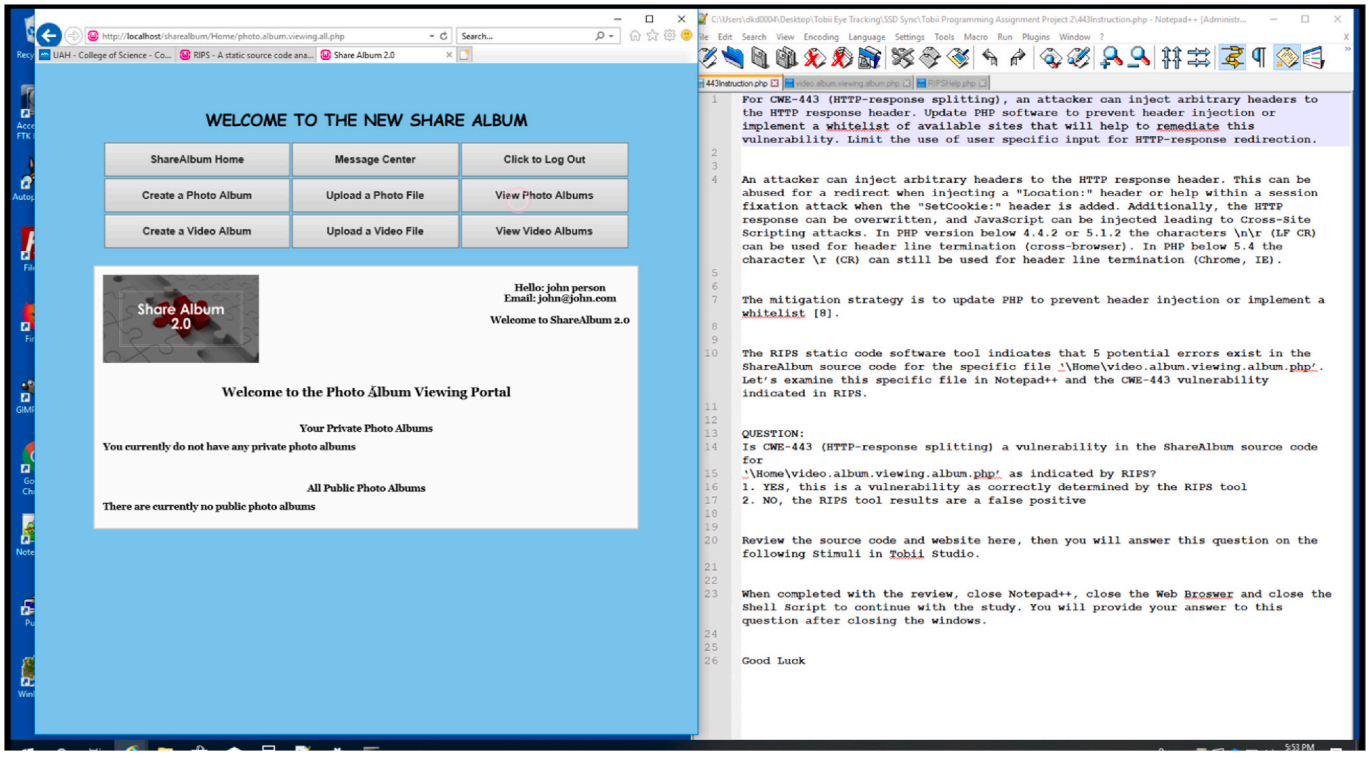


Fig. 2. Split screen between website and instructions/guidance text.

3. Methods

We used the Tobii X2-60 eye tracking hardware and the Tobii Pro Studio for the recording application. The Tobii Pro Studio software provides the ability to present several types of stimuli content, manage participate data collection and provide data filtering of the raw data points. Each student used a single "twenty-four" 1080p monitor to view our stimuli content. Additionally, the X2-60 eye tracker is capable of recording both the left and right pupil sizes of participants [48].

In secure coding tasks, distinct types of source code files, security vulnerabilities, questions and tools are presented in several stages. The programming problems presented in the eye tracking study and in this paper increase in complexity and difficulty. In our experiment, students progressed through our hands-on learning exercises, in which the security weaknesses were easier to find and mitigate was presented first and presented as multiple choice or true/false questions. More challenging security weaknesses was presented afterwards, and participants had to modify actual source code as part of the task. It was beneficial to conduct an actual study to truly elevate visualization methods for eye gaze in secure coding.

The following sections are presented in a similar fashion of increasing complexity. Our eye tracking experiment setup and design had to be able to manage the diverse types of stimuli content presented. Frequently, a participant may have multiple types of stimuli on their computer monitor at once and transition repeatedly between stimuli content of distinct types. An example is shown in Fig. 1 of how participants used both the source code editor and the web browser in a split screen fashion. Fig. 2 provides an example split screen of our photo and video sharing website alongside the instructions guide. Participants switched between multiple source code files and webpages at their own determination. The goal of this approach is to allow participants a natural environment to find and mitigate security weaknesses and at the same time allow the eye tracker to capture their behavior and approaches. Secure coders were asked to keep the split screen view in order to allow for analysis of their behavior. The types of stimuli that we utilized are presented in the following paragraphs. Our visualization solutions to allow us to investigate the behaviors and techniques of participants viewing these types of stimuli are presented in Section 6.

We used a photo and video sharing web-based application. Many of the weaknesses in this application source code are listed in the CWE repository and NVD database [4]. The photo and video sharing web-application consists of approximately thirty files of source code in PHP, HTML, and SQL files with several functions per each file. Additionally, some of the learning content requires our secure coders to utilize third-party security scanning software to help facilitate the secure coders finding of the security flaws.

Our first learning module focused on reading documentation of static stimuli content for our secure coders to gain a general understanding of secure coding flaws with the CWE repository. Furthermore, our first learning module also provided the opportunity to manually review our web-based photo and video sharing project and the associated source code. This manual review of the webpages and files of source code requires utilizing dynamic non-linear eye tracking stimuli content. This permits the students the ability to switch between multiple source code files and scroll through the various code functions defined in the source code of the web-based project. Participants may switch attention between multiple source code functions, source code files as well as other applications such as web pages. Furthermore, our secure coders were given the opportunity to actually write new source code versus simply reading source code. Therefore, this type of stimuli content is actually modified by each coder during the eye tracking recording session. There are various levels of user interactive content that must be examined for secure coders. Our first learning modules requires the secure coder to interact with multiple distinct types of stimuli content to accomplish all the objectives.

Our second learning module allows each secure coder the ability to utilize a static code analysis tool to scan source code to find various security weaknesses. It is recommended by security specialist for secure coders to have the ability to conduct manual code reviews but also utilize security code scanning applications for finding security weaknesses [89–92]. Our two learning modules allow our participants to gain experience with both manual code reviews and static code analysis applications procedures. The Research and Innovation to Promote Security (RIPS) static code analysis tool was selected for our second module [93]. The static code analysis tool adds even more user interactive stimuli content that further complicates the eye tracking analysis.

3.1. Static stimuli content for reading of documentation

Both of our learning modules start with a description of the CWE repository and the benefits it provides to secure coders. Both of our learning modules provided a brief introduction of several CWEs and then provided example source code of several weaknesses and finally provided several demonstration source code snippets that could allow for exploiting a weakness in our application. This content is presented as short paragraphs that are similar to standard reading of a book or research article. Our eye tracking technology recorded the eye gaze of each secure coder while they looked through our learning content. As part of our analysis phase, we explored each coder eye gaze with various visualization methods and common eye tracking metrics to gain an understanding of their reading behavior and patterns of the ordinary documentation paragraphs of text.

3.2. Static stimuli content for reading of source code

An important aspect of our research is to study the behavior and methodologies of secure coders while they perform secure coding tasks. Therefore, we presented each student several CWEs that allowed them to read existing source code functions and determine if a vulnerability existed and how to mitigate the vulnerability. This approach allowed us to introduce coding examples to the participants and present multiple choice or true/false questions to introduce secure coding problems to our participants.

For our first learning module, after reviewing the learning content for CWE-311 [94] and CWE-434 [95], then each student was presented several static screenshot images of snippets of source code and asked which code snippet correctly mitigated the security weaknesses. No scrolling of the source code was allowed for these tasks. This was designed to allow participants that may not be familiar with coding to get a general understanding of reading source code. Additionally, this allowed us to analyze the eye gaze of source code in a similar fashion as the reading of standard document text. Participants provided their answers from two or three multiple choice snippets.

3.3. Dynamic stimuli content for reading of source code

For our second learning module, participants were required to answer if the RIPS tool provided false positives for flagging CWE-73 [96] and CWE-443 [97,98] as potential vulnerabilities. This involved the participants reading source code, potentially using the vulnerability scanner, and answering true or false if each weakness actually existed in the code snipped shown. The second learning module did permit each secure coder the ability to scroll through source code when they are reading source code. During our analysis phase, we explored existing visualizations methods and designed our own visualizations approaches for analyzing eye gazes for secure coders reading dynamic source code stimuli content.

3.4. User interactive stimuli content for patching of vulnerabilities

The ability to read source code and discover weaknesses is a major aspect in secure coding; however, the developers should also have the

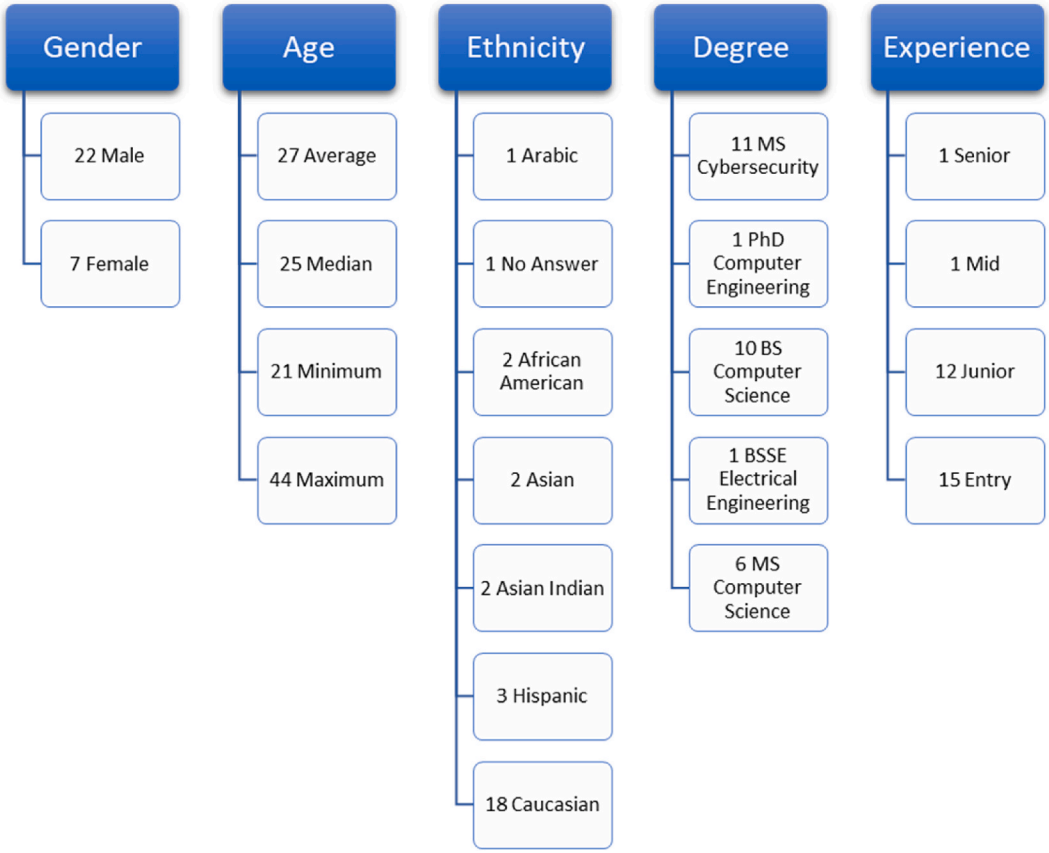


Fig. 3. Manual code analysis learning module participant demographics.

ability to create a mitigation strategy and modify the source code to reduce or eliminate the vulnerability. Therefore, our research also examined participants eye gaze as they write source code to secure our software application. This user interactive stimuli content not only allows participants to control the flow of events by scrolling or switching between files and applications, but additionally it actually allows a participant to modify the stimuli content that is presented. We believe this is a critical aspect to truly capture the behavior of secure coders performing secure coding exercises.

Our first learning module included tasks for the participants to provide programming solutions for CWE-862 [99] and CWE-22 [100]. The students were required to manually scan the source PHP and HTML code files as well as the SQL queries to determine where weaknesses could be exploited. Then, the writing source code tasks involved the participants having the requirement to modify source code to mitigate the security flaw associated with those CWEs. Each participant modified the source code to provide their unique solution to mitigate the security flaw that they discovered in the source code.

For our second learning module, students had to write programming solutions for CWE-79 [101] in addition to using the RIPS tool to analyze the vulnerability report. Students also had the option to use the local web-based photo and video sharing website to determine if weaknesses existed and allows participants to attempt a security exploit. Students were permitted the ability to utilize a search engine or additional websites as needed to further research the CWE or potential solutions. In our analysis phase, we utilized existing eye tracking visualization methods but discovered limitations with existing approaches as the number of AOIs and participants increased for user interactive and user modifiable stimuli content. Therefore, we researched and designed additional visualizations approaches to handle user interactive stimuli content for when each secure coder had the tasks of modifying the stimuli content real-time as it was presented.

4. Data collection

The eye tracking recording sessions (including our first learning module with manual code analysis) had 29 participants; and the eye tracking recording sessions (including our second learning module with a static code analysis tool to provide guidance) had 31 participants. In Fig. 3, for our first learning module, we include the following information about our participants: their identified gender, general information about their age, their identified ethnicity, their current degree, and their current experience level. In Fig. 4, for our second learning module, we include the same information about our participants. Every participant was presented the same learning material about security flaws in source code, and each was presented the same questions in the same order. No participant was ever allowed to see the response or decision-making process of another participant. Each participant was given up to 2 h to work on each learning module with a break in the middle of the recording session. Each learning module was completed on a different day. The average recording duration for the first learning was 49 min and 54 min for the second learning module.

Given that each learning module contained 3 to 4 questions for participants to review and answer and the number of participants that volunteered for our study, our eye tracking design did not include separate groups where some participants received some questions while other participants received a different set of questions. Our research was focused on trying to understand what type of stimuli to present to participants for security coding and what type of visualizations were possible to understand where the participants were looking at. Our finding suggests that which visualization technique to utilize depends upon the stimuli that is presented to participant as well as the question that is asked to the participants. Our current research is focused on finding what type of visualizations are even useful for dynamic non-linear eye tracking stimuli and the benefits and limitations of these

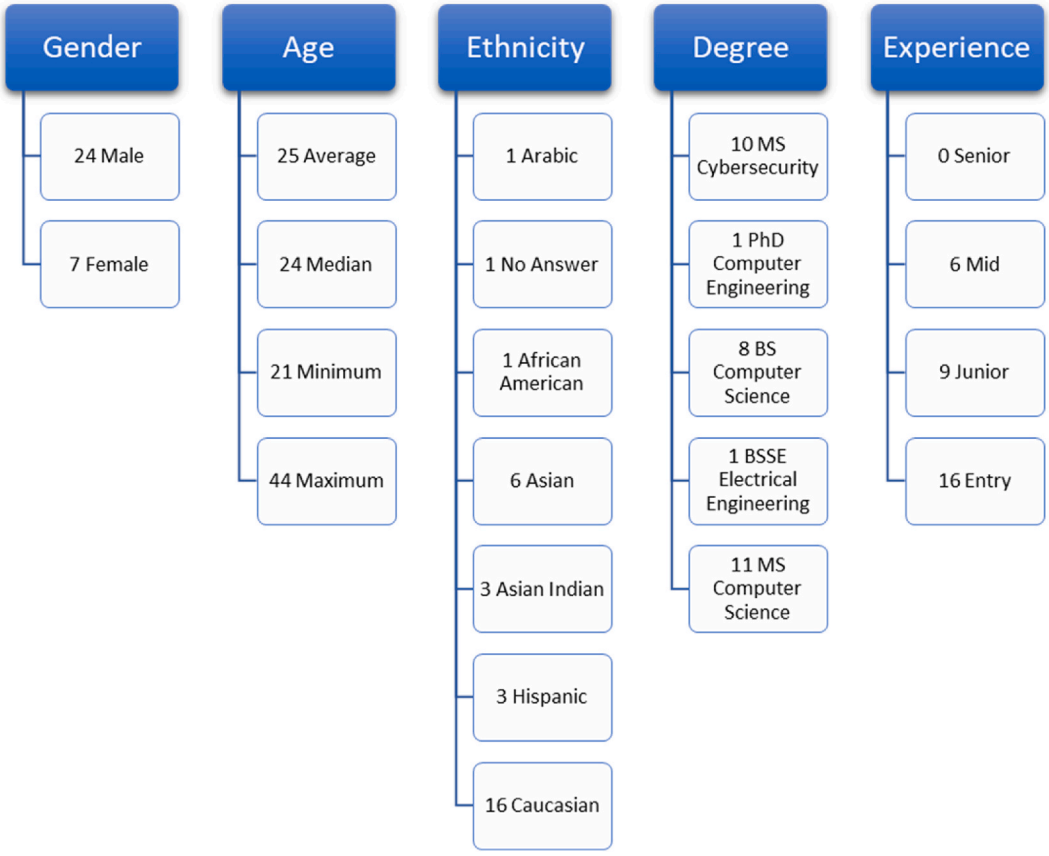


Fig. 4. Static code analysis tool learning module participant demographics.

various visualization methods.

In Figs. 3 and 4, we provide the experience level that participants answered in their questionnaire. For the senior level experience, a participant was expected to have 10 years or more of programming experience. For the mid-level experience, a participant was anticipated to have 5–10 years of programming experience. For the junior-level, we asked that participants mark this category if they were between 3 and 5 years of experience. For the entry-level, we recorded this was 0–3 years of experience.

5. Data processing

A critical step in processing is the creation of AOIs. These steps classify grouping of fixations, AOIs that participants focused on, the transition frequencies, and sequences in moving between stimuli elements or AOIs over time. They allow researchers to examine the patterns and approaches those participants used when approaching problem solving. The creation of AOIs is applicable to multiple types of stimuli. The creation of AOIs is a critical milestone for comparing and contrasting participants' behaviors when they use multiple applications concurrently and transition often between distinct types of stimuli contents.

Most eye tracking software provides various types of filters that can be applied to the raw data to reduce the noise and to classify fixations and saccades [102,103]. We applied Tobii's I-VT Filter to the raw data. This is a commonly applied filter that determines the eye movement classification based upon the velocity of the directional shift of the eye [103,104]. For our application of the filter, we relied upon a threshold of 30° per second velocity [102]. Eye movements below that threshold is marked as fixation and eye gaze data points marked as a saccade if above that threshold.

As part of our processing, we created static and dynamic AOIs to investigate the eye gazes at the application level, source code file or function level as well as the programming statement level. For dynamic AOIs, this was accomplished by advancing over a video recording of a single participant and marking each keyframe. Each keyframe provided the ability to modify existing AOIs or create new AOIs per each participant. This provided the ability to overcome the limitations of some studies that limit the number of lines of source code displayed to one display screen [105]. Our experience with dynamic media and AOIs with keyframes has allowed for more flexibility in the content we present as stimuli; however, it does increase the time required for

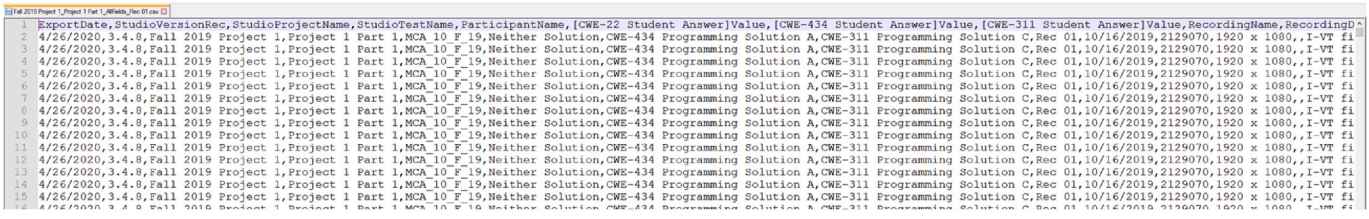


Fig. 5. Example of a raw eye gaze datafile from Tobii Studio.

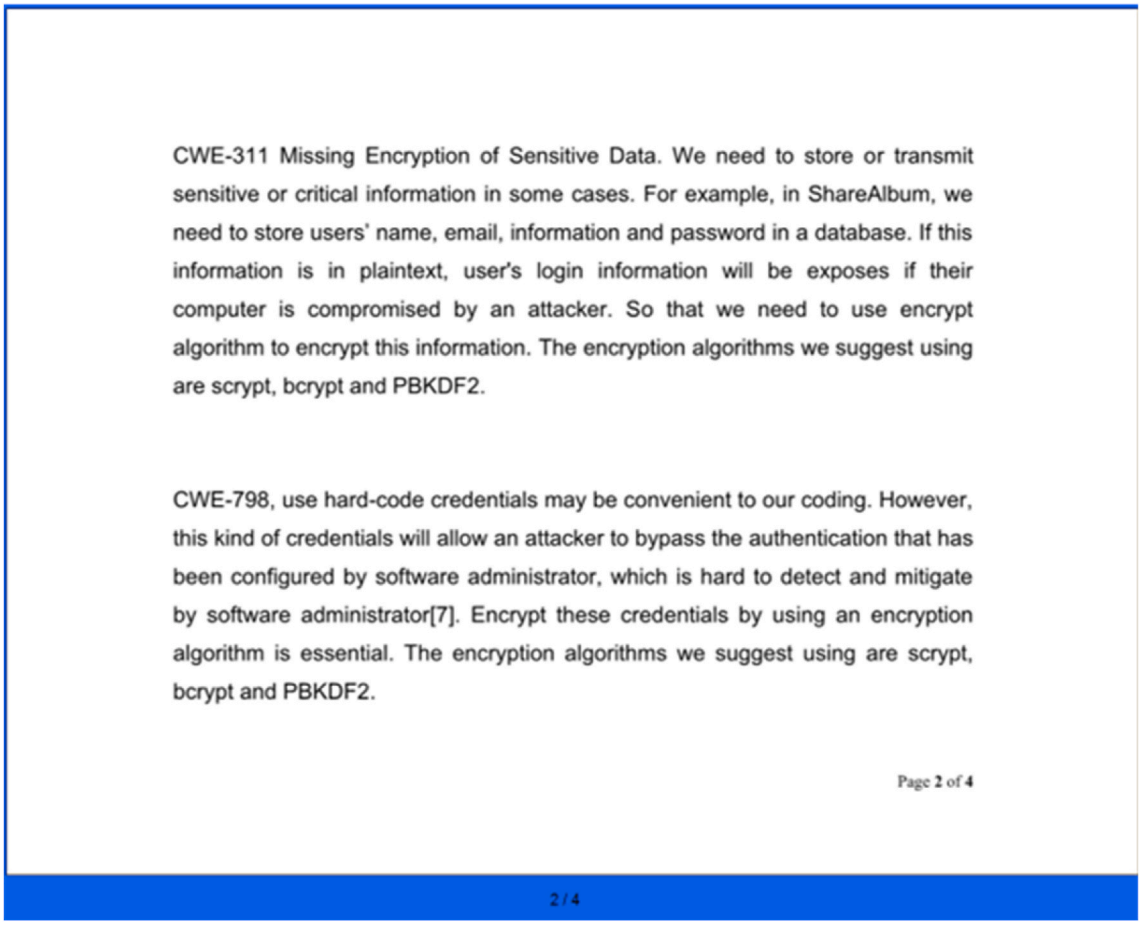


Fig. 6. Example of a Static Stimuli Presented to a Participant using Tobii Studio PDF Stimuli.

processing. After creating the AOIs, then we created our custom data analysis using the Python programming language to create visualizations of our participants fixations, gaze shifts, AOI transitions and behaviors. An example of the raw text-based data is shown in Fig. 5.

6. Data analysis and visualization methodologies

In this section, we present visualization techniques that we utilize to investigate the eye gazes of our secure coders while they performed secure coding activities. This enables us to view eye gazes when participants read documentation, examine multiple source code functions and files, write in source code files, use the web browser, scroll through multiple pages of content. Our approach also examines their transitions between stimuli content and areas of stimuli that are particular interest. Furthermore, this allows us to capture the behaviors of participants viewing these types of stimuli while participants learn and work secure coding problems. Finally, our techniques allow us to compare and contrast participants' strategies to discover and mitigate security weaknesses in source code. The usage of these types of stimuli content, tasks and user interactive control flow of events did increase the complexity of analysis and visualizations. However, this approach allowed us to offer the participants a natural software development and secure code development environment for secure coding.

We start this section with the simplest stimuli content, tasks and analysis approaches and then progress to the approaches used in the dynamic and interactive tasks. First, we present a basic analysis of reading documentation and how to compare multiple students. Then, our analysis progresses to the programming content and the visualizations at a more detailed level of the source code. Afterwards, we present our analysis of user interactive content and split screen applications along with a comparison of several secure coders. Finally, we discuss our visualization techniques for graphically viewing changes in secure coder pupil diameter.

6.1. Analyzing static stimuli of reading content

Our initial methodology focused on analyzing the learning content that is presented to the secure coders. This learning content was presented as stimuli content using Tobii Studio PDF element feature and we did not allow any participant scrolling. This simplified the processing analysis, especially when creating multiple AOIs. AOIs was created for each page of learning content stimuli presented to the participants. An example is shown in Fig. 6 for the learning content associated with CWE-311 and CWE-798 as it was presented to each participant. Fig. 7 contains the same stimuli but shows a single AOI overlay for analysis. Stimuli was either examined with a single AOI overlay as shown in Fig. 7 or multiple AOI overlays. Fig. 8 demonstrates the static stimuli that was utilized for CWE-434 when coders were asked to answer a multiple-choice question. We presented several CWEs that are applicable to the photo and video sharing website to allow participants to learn what coding vulnerabilities may exist in our application. The AOIs and the eye tracking data was then exported to comma separated files per each participant.

Our Python script iterates over the comma separated files and stores the data in a Pandas DataFrame. The script iterates row by row over the data and determines when students switched from one AOI to a different AOI. We down selected to specific AOIs that are specified as input parameters. This allows us to take all of the AOIs for each student and indicate a specific CWE that should be analyzed. Our Python script groups AOIs together and creates a fixation matrix. A fixation matrix does not provide the ability to determine the sequence or transitions

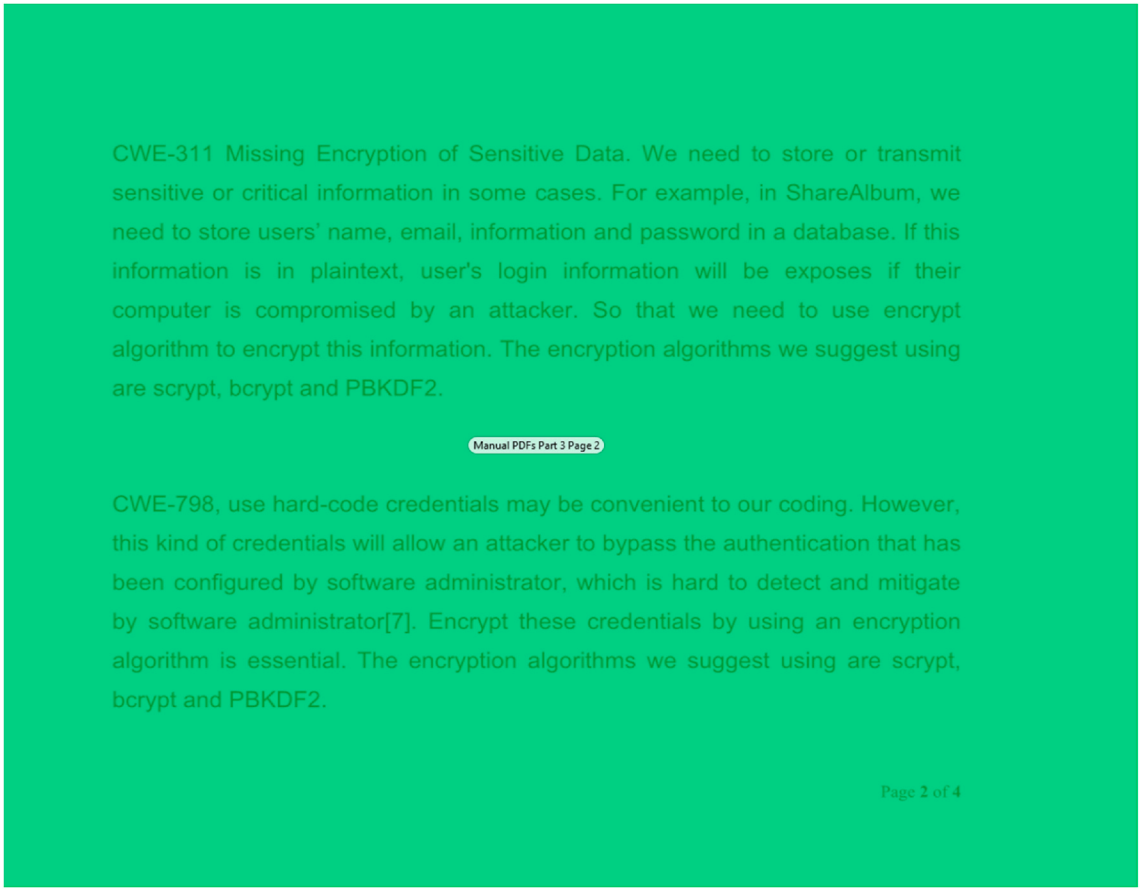


Fig. 7. Example of a static stimuli with overlay for a single AOI for analysis.

between AOIs; however, this output provide an initial high-level overview of the AOIs to determine what secure coders viewed. One can ascertain if some participants have a higher fixation count or fixation duration in some AOIs while ignoring other AOIs. Furthermore, our script uses the AOIs and fixation data to generate various plotting visualizations using the Python package Matplotlib to generate PNG chart images [106]. Our script includes the ability to use Python Multiprocessing capability to allow each participant data processing to operate in a different Python interpreter process. This significantly improved the processing time as the script scales out to the number of threads available on a system and allows for the processing of multiple secure coders at once.

6.2. *Comparing static stimuli of reading stimuli for multiple participants*

One plot chart that we generated is a AOI transition line. This provided the ability to observe objectively the transitions from a AOI to another AOI associated with a specific CWE. An example of four AOIs transition line graphs is displayed in Fig. 9 for two different participants and another two participants presented in Fig. 10 while each were presented questions to CWE-434. The X-axis contains the start time when the participant moved into a specific AOI. The Y-axis contains the AOI unique name. This plot demonstrates the various transitions that each participant took when reading the stimuli content. It can be determined from this diagram that each participant has a unique reading pattern and visible transitions between the various AOIs.

Another plot chart that we generated is a plot of the fixation duration and number of fixations per each AOI over a linear timeline. This provided the ability to observe not only the transition from an AOI to another AOI but also includes the fixation duration totals and number of fixations in each AOI until a transition occurs. This combines the linear nature of eye tracking occurring over a timeline, the transitions associated when moving from one AOI to another AOI as well as an eye tracking metric such as fixation duration or number of fixations.

Prior research has determined that when participants have longer fixation durations then this typically represents deeper cognitive processing than shorter fixation durations [49]. An example is displayed in Fig. 11 for two participants. The X-axis contains the AOI name in the order in which they occur for that specific secure coder and changes when the coder changed AOI. The X-axis is different per participant as different secure coders visit AOIs in the order that they determined viewing the stimuli content. The first X-axis data point is the starting AOI, and the last X-axis data point is the ending AOI. The Y-axis contains the duration of all fixations in that AOI until a transition occurs for the top portion in Fig. 11. The bottom portion in Fig. 11 is for the number of fixations metric. This secure coder had a high number of continuous fixations for Page 1 and Page 2 of the stimuli content associated with CWE-434. The other two pages (Page 3 and Page 4) associated with CWE-434 did not receive as much attention before the secure coder switched to a different page.

We also included the ability to generate data files in the correct format needed to generate the Radial Transition Graph for our eye tracking data as well. We utilized the online graphing capability to generate the visualization [107]. Fig. 12 is an example of using another researcher Radial Transition Graph to visualize our collected data [50,54,56,107]. This can allow one to visualize that Student identified as SCA_112_F_19 only had a duration of 25.5 s in a AOI while Student identified as SCA_104_F_19 had a duration of 1.04 min in that same AOI.

6.3. *Analyzing static stimuli of programming content*

Our initial endeavor focused on creating large single AOIs around the

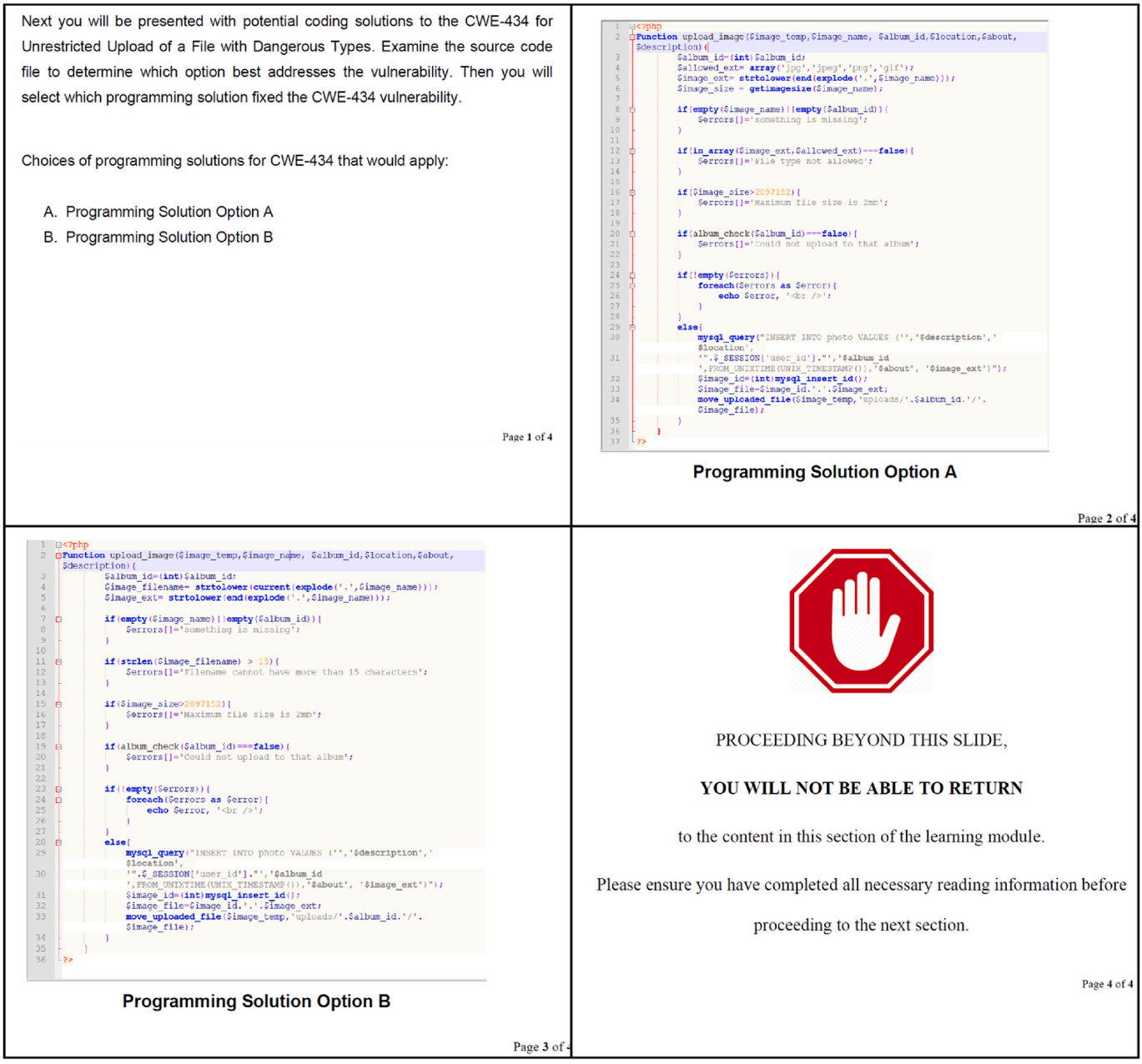


Fig. 8. Static stimuli data content for CWE-434.

entire stimuli content as shown in Fig. 13. Creating a single AOI that contains an entire block of software code may not allow as detailed view as needed to determine what the coder really looked at within a block of software code. Therefore, we created new AOIs on the static pages that had software programming code functions. We created several AOIs that group sections of a software function. An example is shown in Fig. 14 for the questionnaire associated with CWE-434. Our AOIs creation in this more detailed examination of the source code focused on programming logic components that are grouped together to build a programming function. A programming function consists of several key elements including function definition, declaration of variables, programming control statements and programming saving state statements. This analysis technique relied upon the creation of multiple AOIs focused on these more detailed programming statements as shown in Fig. 14. This is different than our prior examination of creating an entire overarching AOI as shown in Fig. 13. This allowed a supplemental detailed analysis of specifically what aspects of a programming function each coder viewed and also the order in which the coder viewed the sections of a programming source code function.

In Fig. 15, our diagram view of the AOIs for a specific secure coder is presented to help show which aspects of a programming function the coder viewed more than other AOIs. We can immediately determine that this specific coder had a greater focus on the AOI associated with the SQL Query execution and the Function Definitions AOI in the first programming option (Option A) than compared to other AOIs. This diagram view provides a quick overview of which AOIs each student viewed more than other AOIs; however, it does not demonstrate the transition or behaviors the coder used when working our secure coding problems. This is a limitation of the fixation matrix that simply summarizes eye tracking statistical results based upon the individually visited stimuli elements or AOIs. No transitions in the stimuli elements or AOIs can be determined from a fixation matrix alone.

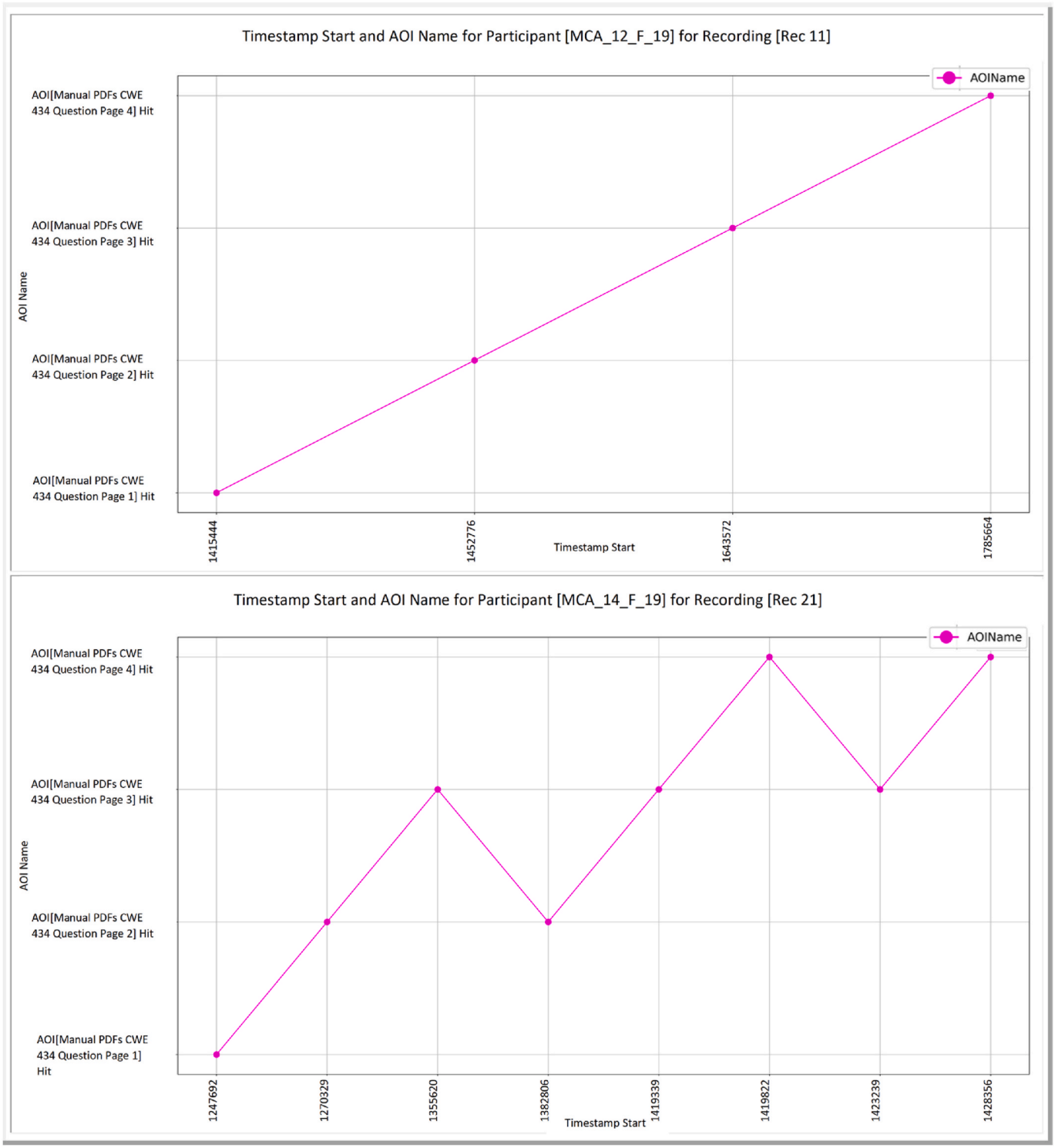


Fig. 9. AOI transition plot for static stimuli learning content.

6.4. *Comparing static stimuli of programming for multiple participants*

We examined the transitions from each AOI over a timeline in order to examine the techniques that secure coders utilized. An example of two participants is presented in Fig. 16, followed by two additional participants in Fig. 17 and finally two more participants in Fig. 18. The X-axis contains the start time when the participant moved into the AOI. The Y-axis contains the AOI unique name for both programming option A and programming option B.

In the top half of Fig. 16, we determined that the coder viewed several AOIs in programming option A before transitioning to programming option B. The participant also had several transitions back to option A while viewing option B. In the bottom half of Fig. 16, we can establish that another participant viewed several AOIs in programming option A before going to option B. However, after a few transitions in option B, the participant went back to option A and performed twenty-eight transitions within option A before returning to option B for a second viewing.

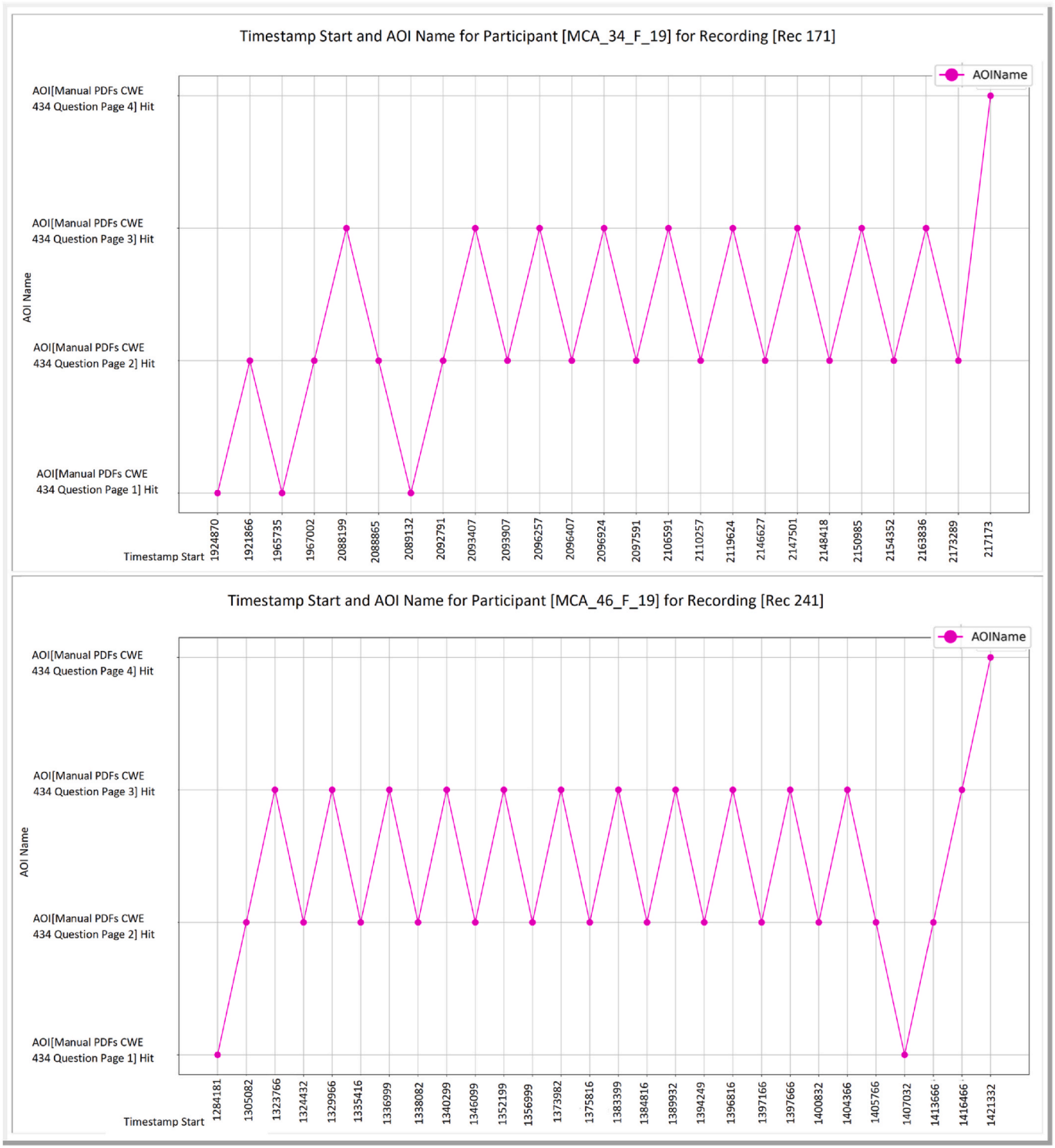


Fig. 10. AOI transition plot for static stimuli learning content.

The top half of Fig. 17 demonstrates that a third coder had several transitions between the function definition and the declaration of variables. This chart also indicates that the third participant mostly focused on one option at a time and made few transitions between the two options. In the bottom half of Fig. 17, we can determine that the transition pattern of a fourth secure coder appears to transition between the two options more rapidly than our prior participants.

In the top half of Fig. 18, we can ascertain that a fifth participant examined option A for a great amount of time before transitioning to option B. We can also determine that the participant had several transitions between the AOI associated with a programming check to determine if input fields are missing and a check for incorrect file extension. In the bottom half of Fig. 18, we can determine a sixth participant viewed option A before transitioning over to option B. Then, the coder reviewed both programming options. The data indicates that the sixth participant had several transitions between the function definition AOI and the variable declaration AOI along with the AOI associated with checking for incorrect file extension.

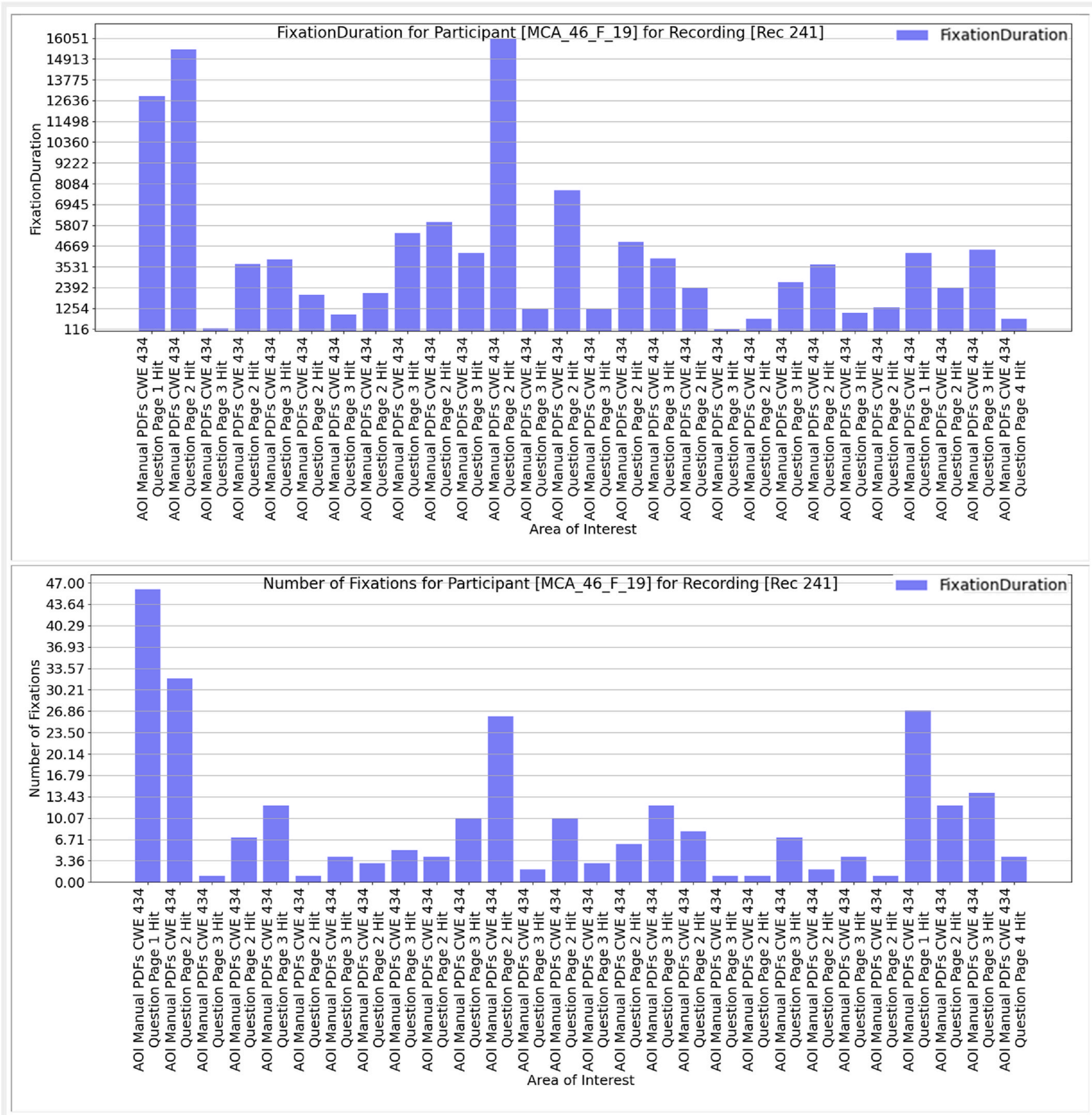


Fig. 11. Fixation durations and number of fixations plot for static stimuli.

For CWE-434, the programming solution that contained the security flaw was located in programming option A. This approach to analyzing the AOI transitions allows researchers to establish a transition pattern and revisit pattern of participants working a problem. We also examined the fixation duration and number of fixations for these same participants working problem for CWE-434. This view allows for examining which AOIs the participants focused more in but also when those fixations occurred over a timeline. This is not possible with the fixation matrix that simply provides an overall summary of fixation count or fixation duration per each stimuli content or AOI. This demonstrates that some secure coders have more fixations in the same programming option before switching to a different programming option while other secure coders have fixations stretched throughout the programming options when working a secure coding problem. This type of view demonstrates each participant's return or revisit to a previous stimuli content element or AOI showing both the count of revisits as well as the order of revisits [49]. It can be determined from that each secure coder has a unique reading pattern and the order of transition between the various AOIs can be visually analyzed per each participant.

6.5. Analyzing multiple user interactive stimuli content

The previous analysis methods have focused on static stimuli content that did not allow any secure coder participating in the eye tracking experiment to scroll in the text. The inability to scroll restriction allowed for simpler post-processing of the eye tracking data. This is a common

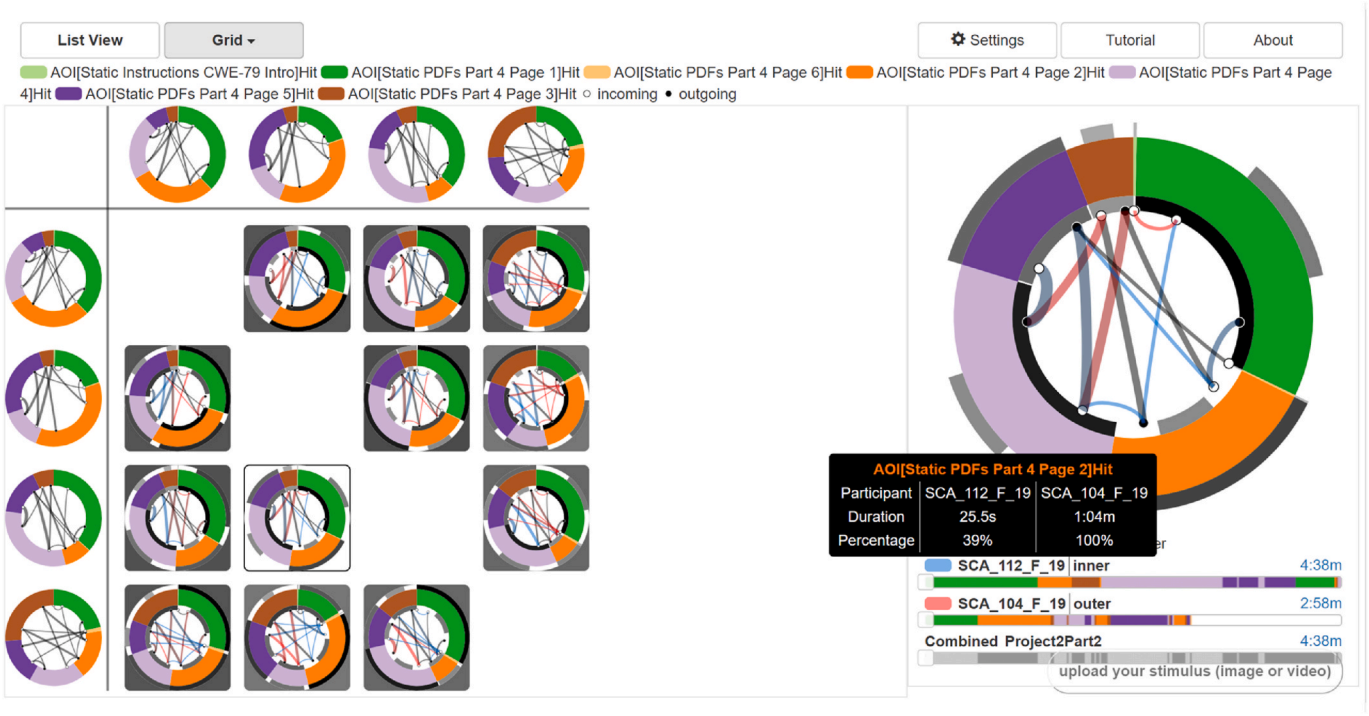


Fig. 12. Comparison of four participants for CWE-79 using the Radial Transition Graph.

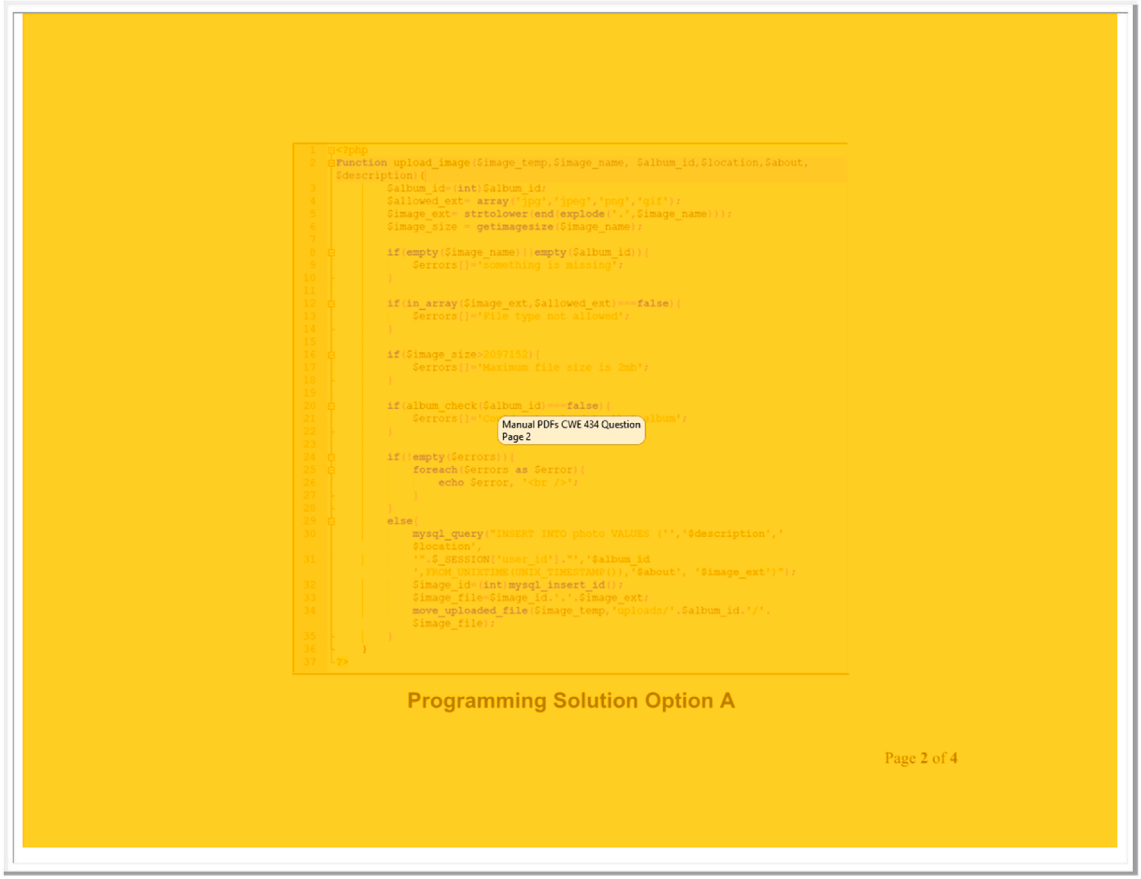
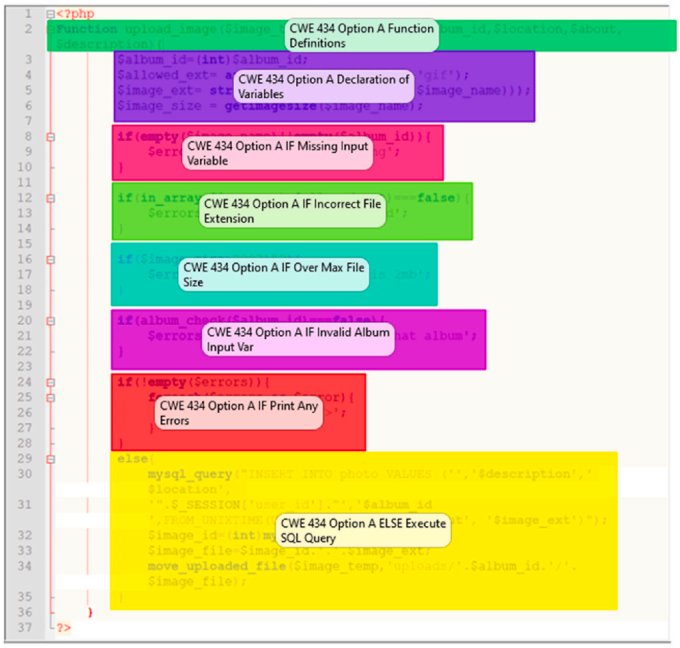


Fig. 13. Example from CWE-434 of creating a Single AOI for Software Code for Static Stimuli.



Programming Solution Option A

Page **2** of **4**

Fig. 14. Example from CWE-434 of creating Multiple AOIs for Software Code for Static Stimuli.

Student MCA_18_F_19 for Recording Rec 51

AOI Name	Number of Fixations	Fixation Durations	Total Duration Recording
AOI[CWE 434 Option A Function Definitions]Hit	18	4675.0	11169.0
AOI[CWE 434 Option A Declaration of Variables]Hit	10	2782.0	13353.0
AOI[CWE 434 Option A IF Missing Input Variable]Hit	2	303.0	1870.0
AOI[CWE 434 Option A IF Incorrect File Extension]Hit	6	1601.0	8833.0
AOI[CWE 434 Option A IF Invalid Album Input Var]Hit	2	400.0	683.0
AOI[CWE 434 Option A IF Over Max File Size]Hit	1	233.0	3100.0
AOI[CWE 434 Option A ELSE Execute SQL Query]Hit	36	12961.0	30857.0
AOI[CWE 434 Option A IF Print Any Errors]Hit	6	1482.0	3017.0
AOI[CWE 434 Option B Function Definitions]Hit	4	717.0	7384.0
AOI[CWE 434 Option B IF Missing Input Variable]Hit	1	267.0	1403.0
AOI[CWE 434 Option B IF Incorrect File Extension]Hit	7	1383.0	9536.0
AOI[CWE 434 Option B IF Over Max File Size]Hit	3	667.0	2266.0
AOI[CWE 434 Option B IF Invalid Album Input Var]Hit	7	1533.0	5634.0
AOI[CWE 434 Option B IF Print Any Errors]Hit	1	417.0	1217.0
AOI[CWE 434 Option B ELSE Execute SQL Query]Hit	2	650.0	7101.0
AOI[CWE 434 Option B Declaration of Variables]Hit	6	1668.0	9837.0

AOI Name	Number of Fixations	Fixation Durations	Total Duration Recording
AOI[CWE 434 Option A Function Definitions]Hit	0.16071428571428573	0.1472951258703803	0.09524987207914037
AOI[CWE 434 Option A Declaration of Variables]Hit	0.08928571428571429	0.0876524150099247	0.1138751492410029
AOI[CWE 434 Option A IF Missing Input Variable]Hit	0.017857142857142856	0.009546614575128391	0.01594746716697936
AOI[CWE 434 Option A IF Incorrect File Extension]Hit	0.05357142857142857	0.050442673052081034	0.07532833020637898
AOI[CWE 434 Option A IF Invalid Album Input Var]Hit	0.017857142857142856	0.012602791518321308	0.005824663141736313
AOI[CWE 434 Option A IF Over Max File Size]Hit	0.008928571428571428	0.007341126059422162	0.026436977656489852
AOI[CWE 434 Option A ELSE Execute SQL Query]Hit	0.32142857142857145	0.40836195217240062	0.26315026436977657
AOI[CWE 434 Option A IF Print Any Errors]Hit	0.05357142857142857	0.046693342575380443	0.025729148899880608
AOI[CWE 434 Option B Function Definitions]Hit	0.03571428571428571	0.022590503796590944	0.06297117516629712
AOI[CWE 434 Option B IF Missing Input Variable]Hit	0.008928571428571428	0.008412363338479473	0.011964864403888794
AOI[CWE 434 Option B IF Incorrect File Extension]Hit	0.0625	0.043574151674595926	0.08132355449428862
AOI[CWE 434 Option B IF Over Max File Size]Hit	0.026785714285714284	0.02101515485680078	0.019324577861163227
AOI[CWE 434 Option B IF Invalid Album Input Var]Hit	0.0625	0.04830019849396641	0.04804707487634317
AOI[CWE 434 Option B IF Print Any Errors]Hit	0.008928571428571428	0.013138410157849964	0.010378645744499404
AOI[CWE 434 Option B ELSE Execute SQL Query]Hit	0.017857142857142856	0.020479536217272127	0.06055773494797885
AOI[CWE 434 Option B Declaration of Variables]Hit	0.05357142857142857	0.052553640631399855	0.08389049974415828

Fig. 15. Static PDF Stimuli Data Analysis for CWE-434 with a detailed AOI.

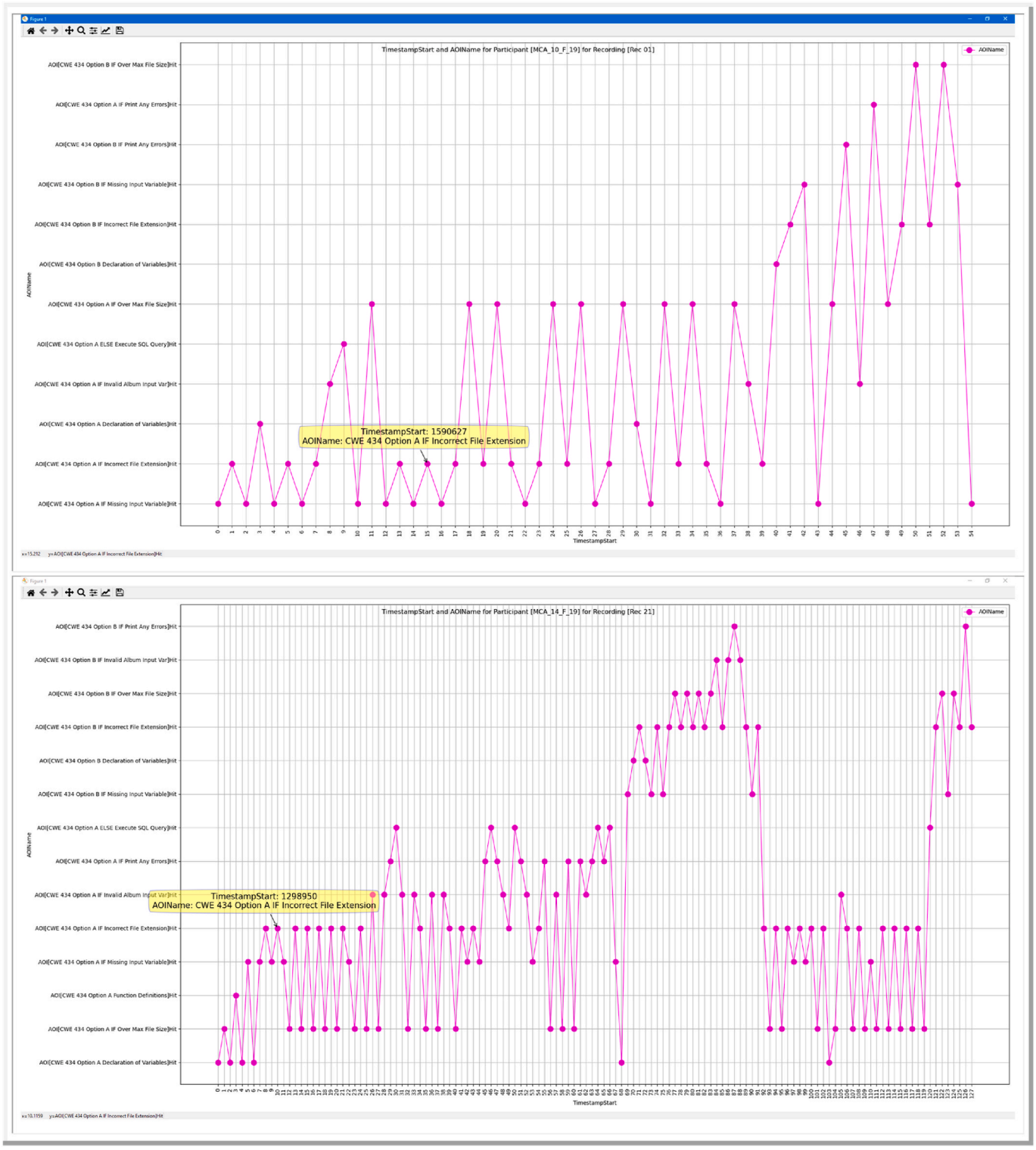


Fig. 16. AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI.

restriction that researchers have faced with eye tracking research [48, 79]. However, we believe that this also restricts the ability of the participant to follow a natural flow of problem solving that is typical for reading and writing software program code [105]. Therefore, our experiments also included dynamic interactive stimuli in both of our learning modules. The post-processing data analysis is more complicated and time consuming for dynamic user interactive stimuli content than compared to static stimuli; however, our belief is that it is necessary aspect of eye tracking and software development.

The eye tracking software application that we utilized allows for re-watching of the dynamic interactive stimuli as a video recording for each secure coder. Each secure coder raw datapoints are saved with a video recorded file that contains the stimuli content and furthermore metadata for the eye gazes in relationship to the video frames. We used the Tobii Studio application to create, move, transform, and disable AOIs as needed for each coder for the dynamic content. This was

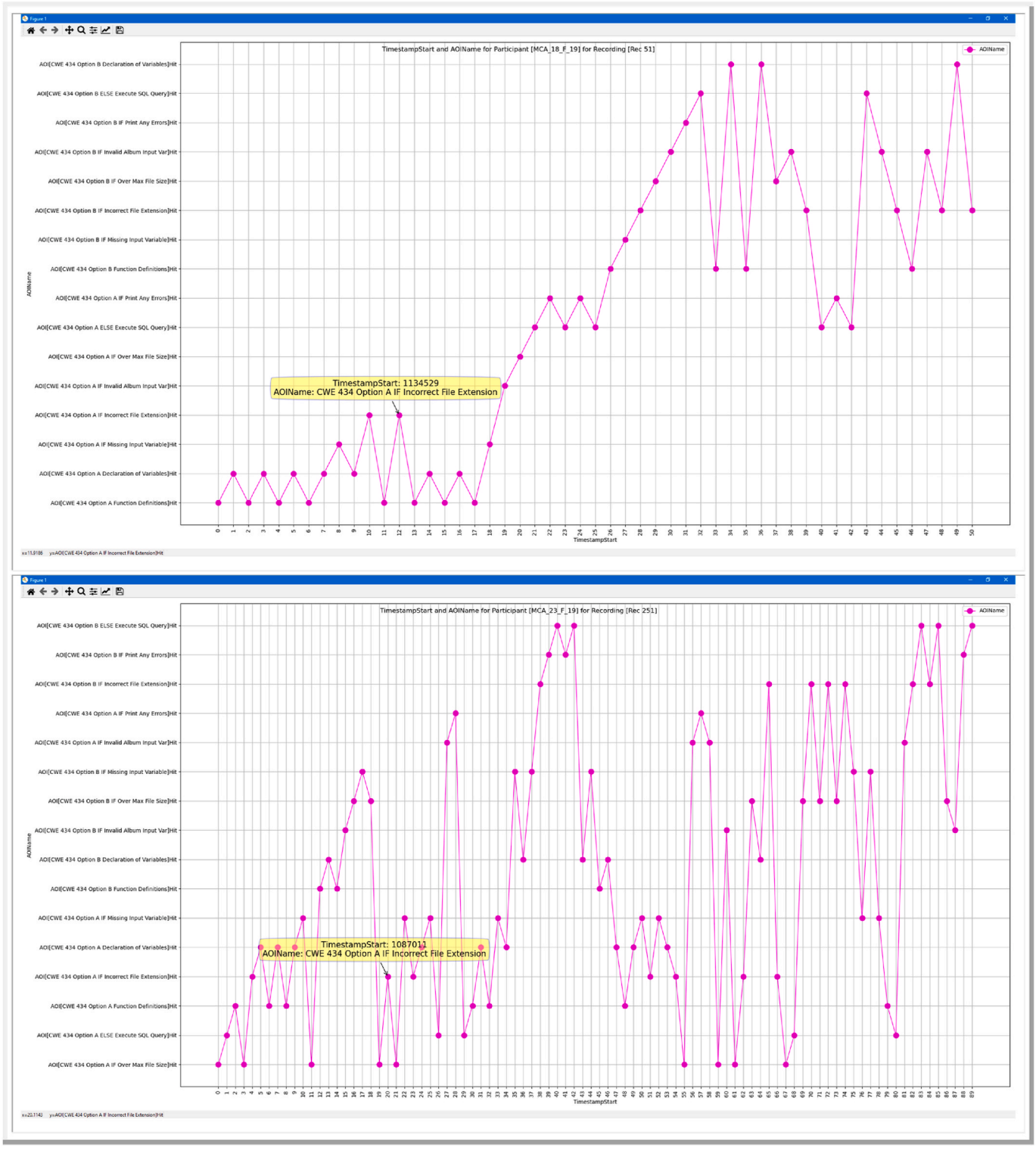


Fig. 17. AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI.

accomplished by advancing the video recording at an interval of 0.2 s for each coder and annotating AOIs for each timestep. Depending upon where the participant scrolled and which source files or websites the coder viewed, then that altered which AOIs were active and the location of the AOIs. The AOIs for the dynamic interactive content for then exported from Tobii Studio for further data analysis. For our dynamic stimuli content, particularly the source code containing multiple code files with several programming functions per each file, our AOI creation was focused on the function level due to the considerable number of AOIs that was required. This entailed the creation as few as 10 AOIs to as high as 120 AOIs depending upon the student and the specific CWE. The dynamic interactive stimuli significantly increased the number of AOIs that was generated to cover all participants and revealed some shortcoming of our initial analysis approaches when using more limited static stimuli content.

Our first investigation was with the fixation matrix and the transition

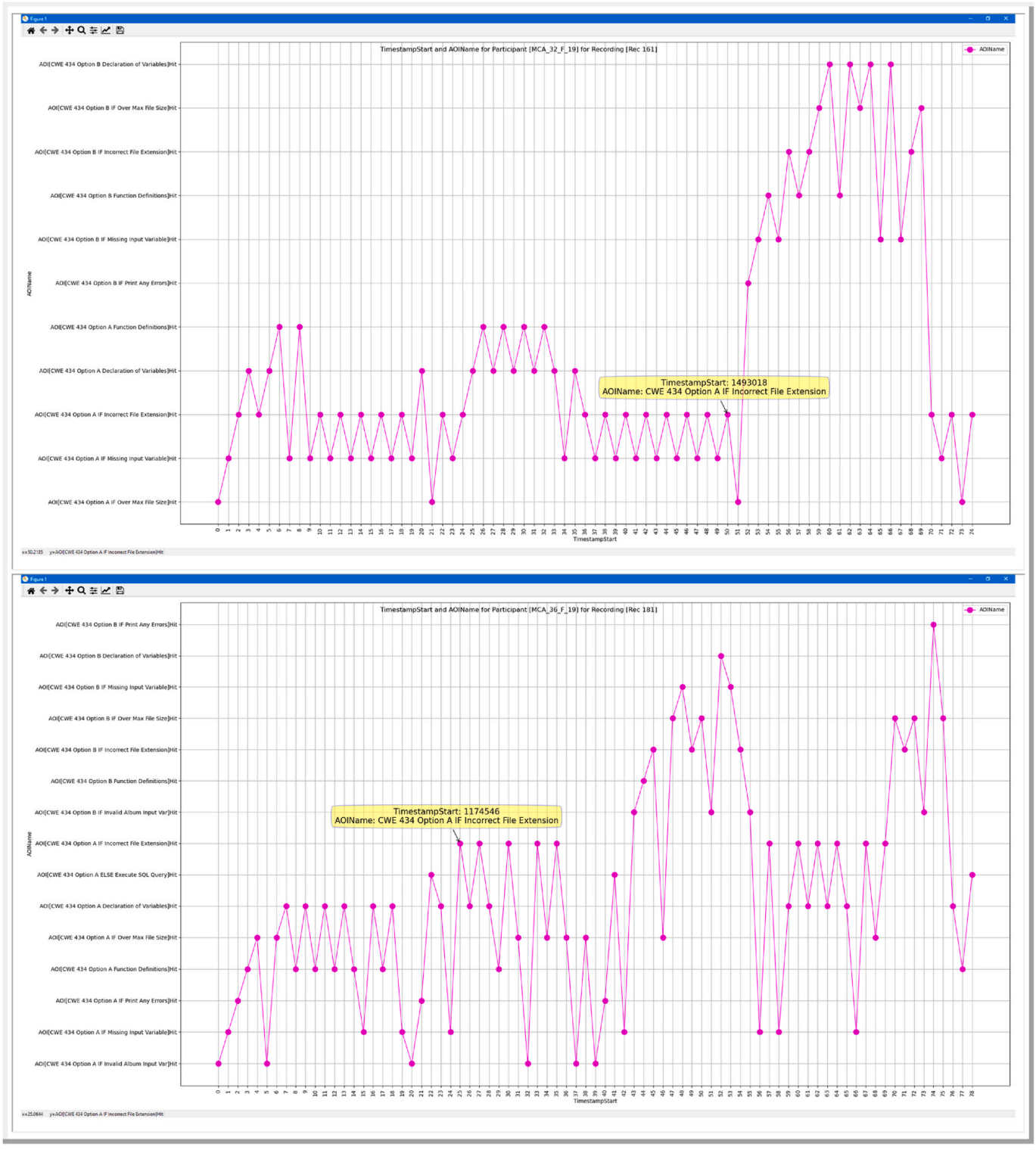


Fig. 18. AOI Transition Plot for Static Stimuli for CWE-434 with a multiple AOI.

plot similar to our examination of the static stimuli content. As shown in Fig. 19, it can be visually verified that the secure coder focused significantly more on the AOI associated with the source code function for getting notifications and another AOI associated with the source code function for sending notifications. The security vulnerability for CWE-79 did occur in the AOI for sending notification. We can also see when and how often the specific secure coder utilized the RIPS static code analysis tool as well as the Google search engine to lookup additional information. As can be visually established from Fig. 19, as the number of AOIs increases, then the analysis of the transitions becomes more difficult to visualize and develop meaningful results.

6.6. *Comparing user interactive stimuli of multiple participants*

Our swimlane approach was developed to visually determine the transitions between various high-level components and specific AOIs.

Student SCA_104_F_19 for Recording Rec 322

AOI Name	Number of Fixations	Fixation Durations	Total Duration Recording
AOI[79 Instructions]Hit	48	6553.0	80770.0
AOI[79 Web ShareAlbum Home Page]Hit	11	2351.0	36166.0
AOI[79 Notify File Function Include Headers]Hit	13	3083.0	11951.0
AOI[79 Web RIPS Home Page]Hit	33	8368.0	23577.0
AOI[79 Notify File Function Check User Logged In]Hit	23	6185.0	28462.0
AOI[79 Notify File Function Welcome Message]Hit	16	5833.0	12601.0
AOI[79 Notify File Function Get User Notifications]Hit	164	31398.0	125828.0
AOI[79 Instructions RIPS Help]Hit	7	3216.0	6717.0
AOI[79 Web RIPS Scan Results Popup]Hit	7	1784.0	25497.0
AOI[79 Web RIPS Scan Results Issue 54]Hit	19	4236.0	25319.0
AOI[79 Web RIPS Scan Results Issue 1]Hit	6	1967.0	5117.0
AOI[79 Web RIPS Scan Results Issue 55]Hit	19	2571.0	29948.0
AOI[79 Web RIPS Scan Results Issue 54 Code Popup]Hit	8	2167.0	7433.0
AOI[79 Web RIPS Scan Results Issue 56]Hit	6	866.0	7233.0
AOI[79 Web RIPS Scan Results Issue 55 Code Popup]Hit	4	649.0	1467.0
AOI[79 Web RIPS Scan Results Issue 57]Hit	6	1985.0	7620.0
AOI[79 Web RIPS Scan Results Issue 59]Hit	5	1050.0	97895.0
AOI[79 Web RIPS Scan Results Issue 58]Hit	4	799.0	5729.0
AOI[79 Web RIPS Scan Results Issue 60]Hit	34	7051.0	21806.0
AOI[79 Web RIPS Scan Results Issue 61]Hit	11	1952.0	26347.0
AOI[79 Web RIPS Scan Results Issue 63]Hit	6	1335.0	6800.0
AOI[79 Web RIPS Scan Results Issue 62]Hit	16	2631.0	16802.0
AOI[79 Web RIPS Scan Results Issue 62 Code Popup]Hit	1	183.0	5817.0
AOI[79 Web RIPS Scan Results Issue 66]Hit	16	2535.0	20731.0
AOI[79 Web RIPS Scan Results Issue 65]Hit	7	1650.0	13814.0
AOI[79 Web RIPS Scan Results Issue 64]Hit	4	1016.0	6466.0
AOI[79 Notify File Find Box Popup]Hit	13	3835.0	15618.0
AOI[79 Web Google Search]Hit	115	32502.0	159475.0
AOI[79 Notify File Function Send Notification to Another User]Hit	104	22537.0	97288.0

AOI Name	Number of Fixations	Fixation Durations	Total Duration Recording
AOI[79 Instructions]Hit	0.06611570247933884	0.04037883269249729	0.08682201540588244
AOI[79 Web ShareAlbum Home Page]Hit	0.015151515151515152	0.014486591738144533	0.03887588224797752
AOI[79 Notify File Function Include Headers]Hit	0.01790633608815427	0.018997091590259294	0.012846476490227821
AOI[79 Web RIPS Home Page]Hit	0.045454545454545456	0.051562654047126094	0.025343601055150308
AOI[79 Notify File Function Check User Logged In]Hit	0.03168044077134986	0.03811125899635216	0.030594629224739708
AOI[79 Notify File Function Welcome Message]Hit	0.02203856749311295	0.035942275460909	0.013545180340838488
AOI[79 Notify File Function Get User Notifications]Hit	0.22589531680440772	0.19347086660751256	0.1352561663302139
AOI[79 Instructions RIPS Help]Hit	0.009641873278236915	0.01981662230109435	0.007220298099310541
AOI[79 Web RIPS Scan Results Popup]Hit	0.009641873278236915	0.010992802918268757	0.027407464736954126
AOI[79 Web RIPS Scan Results Issue 54]Hit	0.026170798898071626	0.026101745045844425	0.027216127374786896
AOI[79 Web RIPS Scan Results Issue 1]Hit	0.008264462809917356	0.012120427881297446	0.00550041169780736
AOI[79 Web RIPS Scan Results Issue 55]Hit	0.026170798898071626	0.01584220644779651	0.032191973720135 78
AOI[79 Web RIPS Scan Results Issue 54 Code Popup]Hit	0.011019283746556474	0.01335280489007197	0.007989947263983214
AOI[79 Web RIPS Scan Results Issue 56]Hit	0.008264462809917356	0.00533619244799369	0.007774961463795316
AOI[79 Web RIPS Scan Results Issue 55 Code Popup]Hit	0.005509641873278237	0.003999063393473332	0.001576920844378229
AOI[79 Web RIPS Scan Results Issue 57]Hit	0.008264462809917356	0.012231341812087154	0.008190958987158898
AOI[79 Web RIPS Scan Results Issue 59]Hit	0.006887052341597796	0.006469979296066253	0.10523017454697117
AOI[79 Web RIPS Scan Results Issue 58]Hit	0.005509641873278237	0.0049233461500542 25	0.006158268246382327
AOI[79 Web RIPS Scan Results Issue 60]Hit	0.046831955922865015	0.04344745144434586	0.023439901794486476
AOI[79 Web RIPS Scan Results Issue 61]Hit	0.015151515151515152	0.012027999605639357	0.02832115438775269
AOI[79 Web RIPS Scan Results Issue 63]Hit	0.008264462809917356	0.008226116533569949	0.007309517206388518
AOI[79 Web RIPS Scan Results Issue 62]Hit	0.02203856749311295	0.016211919550428867	0.018060957073785276
AOI[79 Web RIPS Scan Results Issue 62 Code Popup]Hit	0.0013774104683195593	0.0011276249630286898	0.006252861998465001
AOI[79 Web RIPS Scan Results Issue 66]Hit	0.02203856749311295	0.015620378586217095	0.022284353118476524
AOI[79 Web RIPS Scan Results Issue 65]Hit	0.009641873278236915	0.010167110322389825	0.014849069218978087
AOI[79 Web RIPS Scan Results Issue 64]Hit	0.005509641873278237	0.0062604752045745836	0.006950490920074729
AOI[79 Notify File Find Box Popup]Hit	0.01790633608815427	0.023630829143251503	0.016788241136672922
AOI[79 Web Google Search]Hit	0.1584022038567493	0.20027358769594794	0.17142430242482484
AOI[79 Notify File Function Send Notification to Another User]Hit	0.14325068870523416	0.13887040323375727	0.1045776926434009

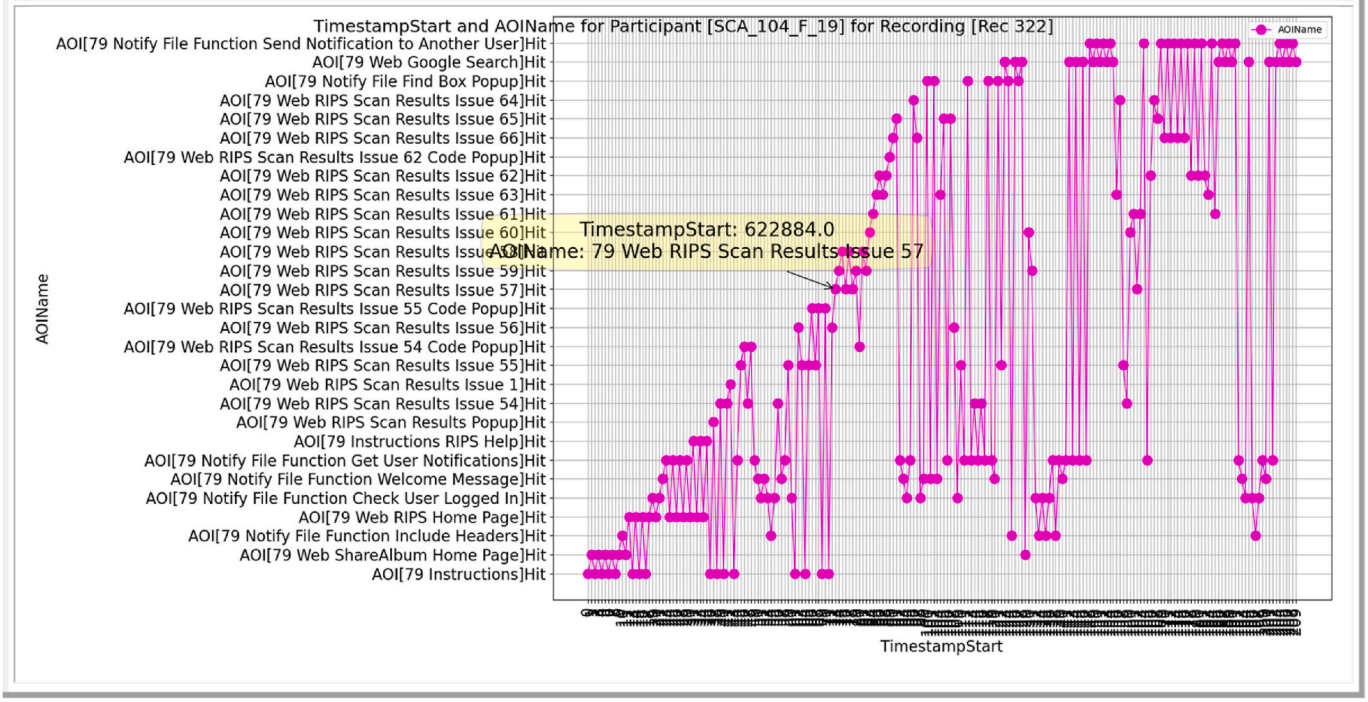


Fig. 19. Example of fixation matrix and transition plot for CWE-79.

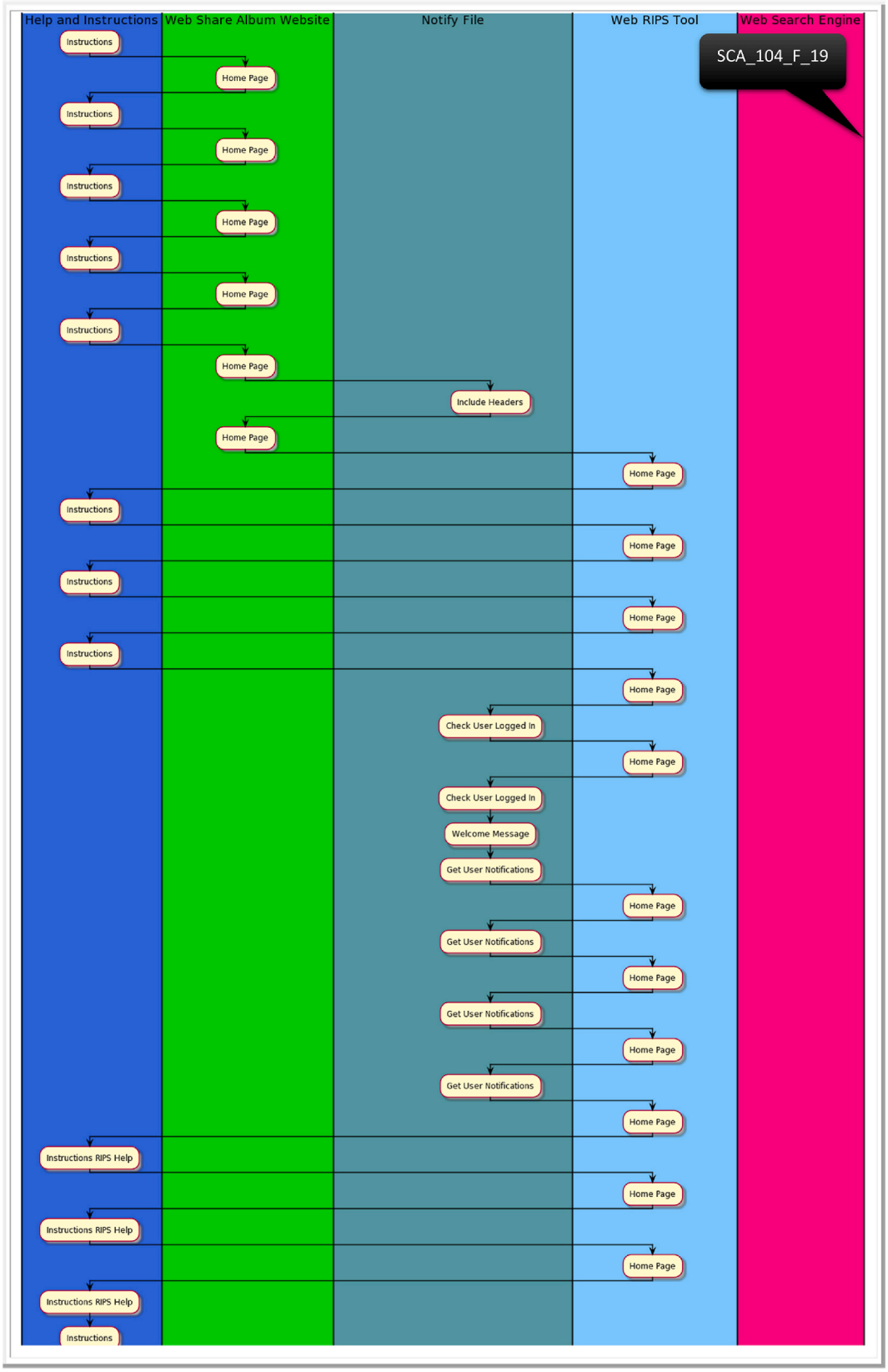


Fig. 20. PlantUML swimlane representation.

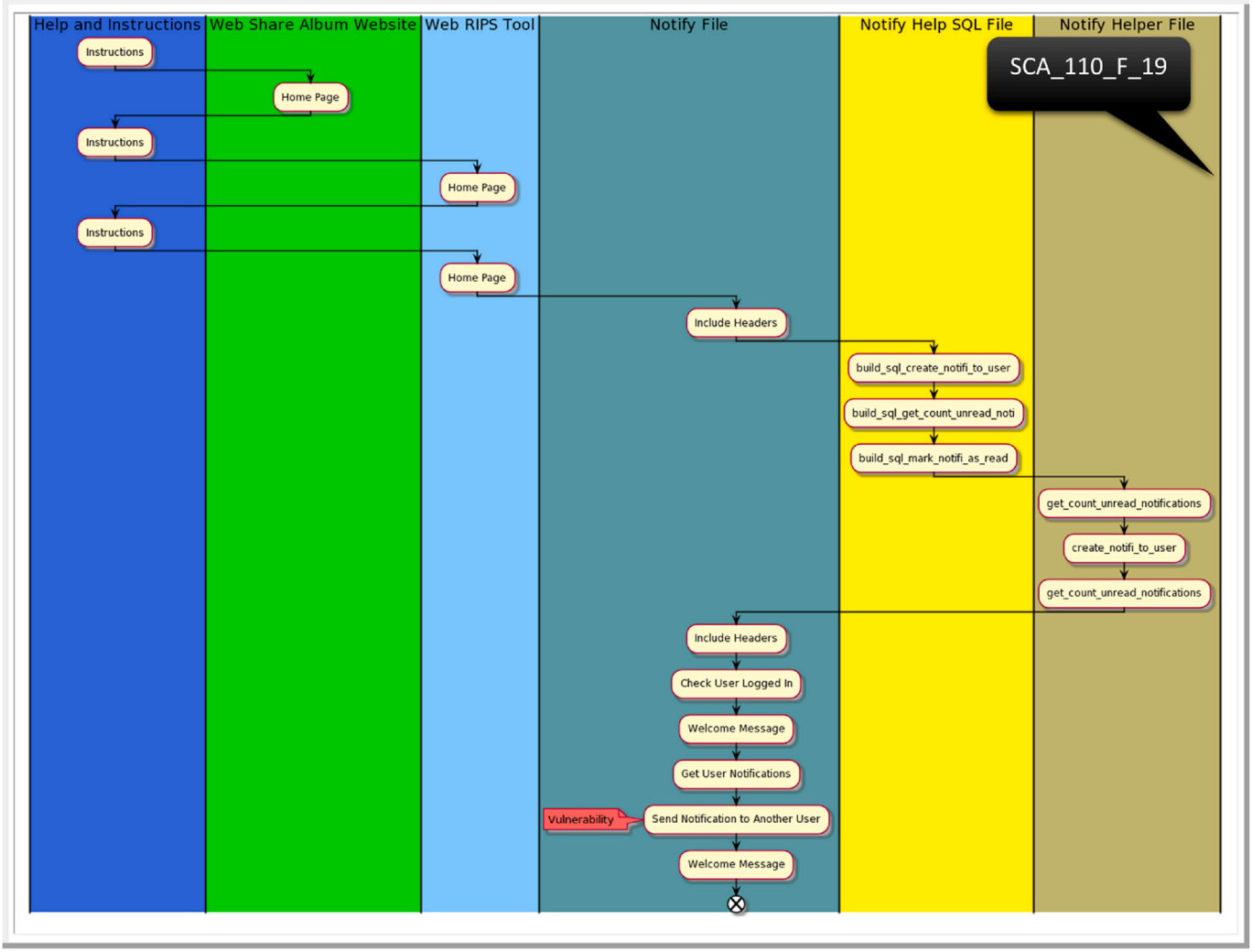


Fig. 21. PlantUML swimlane representation.

This allowed us to group AOIs together into categories that have common source files or usages. For example, we grouped AOIs associated with instructions together and AOIs associated with using the static code analysis tool. Then, AOIs for each source code function contained in the same source file was also grouped together. The swimlane diagram approach allows us to clearly recognize the differences in the multiple stimuli and applications that each secure coder utilized when working the secure coding learning module. The transitions between the multiple stimuli content can also be easily visualized with swimlane diagrams. Additionally, this diagramming technique allows for comparison of multiple participants and visually ascertain their behavior.

PlantUML allowed for the generation of swimlane diagrams using an automated script processing approach from text strings [108]. This removed the need to create the swim lane diagrams manually for each secure coder. An example of one participant can be viewed in Fig. 20 and an example of a second participant can be visualized in Fig. 21. The swimlane diagram provides a visualization method of grouping AOIs together to help visualize when the coders make a more defined transition between various components such as source code, tools, instructions, and web searching. However, a limitation of the swimlane diagram is that often all of the swimlanes cannot fit on a typical single sheet of paper. The PlantUML web server provides the capability of generating Scalable Vector Graphics (SVG) of the data to allow for better viewing of swimlanes that contain significant transitions between stimuli components [108].

A transition matrix can provide an analyst with a high-level overview summary of the transitions between stimuli content or AOIs. Our transition matrix provides a legend for a shorter AOI unique identifier that is matched to a specific AOI name. This provides for easier readability in the matrix as the number of AOIs increased. The rows in the matrix represent the current AOI and the column represent the next AOI that was transitioned to after leaving the current AOI. We provided a matrix diagram representing the transition count as well as the transition ratio for each secure coder. In Fig. 22, we can visually determine for one secure developer that for the AOI Web RIPS Tool Home Page the only transition is to the Instruction AOI. Similarly, we can determine that the same developer only transition from the Web Search Engine Google AOI to another AOI is the AOI associated with the send notification to another user function located in the notify file source code. The cyber vulnerability for CWE-79 existed inside of the send notification to another user function. A transition matrix provides an overall summary of transitions that allows for comparing multiple participants.

The transition matrix approach provides the ability to view the transitions per each stimuli element or AOI and the swimlane diagrams also provide that ability but allows for grouping of specific AOIs together. We wanted to explore additional diagramming methods to visualize the data with a different diagram but maintain the grouping components selected in the swimlane diagram. Therefore, we created a state transition diagram that uses the PlantUML state diagramming. Directly using all of the transitions from all of the AOIs may result a

CWE 79 Legend

AOI ID	AOI Name
A	Instructions
B	Instructions RIPS Help
C	Notify File Find Box Popup
D	Notify File Check User Logged In
E	Notify File Get User Notifications
F	Notify File Include Headers
G	Notify File Send Notification to Another User
H	Notify File Welcome Message
J	Notify Help File create_notifi_to_user
T	Web RIPS Tool Home Page
S	Web Search Engine Web Google Search
AG	Web RIPS Tool Scan Results Issue 7
AH	Web RIPS Tool Scan Results Issue 8

Student SCA_126_F_19 for Recording Rec 462

AOI	A	B	F	H	J	T	D	E	G	C	AG	S	AH
A	-	1	1	1	-	-	-	-	-	-	-	-	-
B	-	-	-	-	1	-	-	-	-	-	-	-	-
J	-	-	-	-	-	1	-	-	-	-	-	-	-
T	1	-	-	-	-	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	1	-	-	-	-	-	-
D	-	-	-	2	-	-	-	1	-	-	-	-	-
H	-	-	-	-	-	-	-	8	1	-	-	-	-
E	1	-	-	6	-	-	2	-	3	2	-	-	-
C	-	-	-	-	-	-	-	-	3	-	4	-	-
G	-	-	-	-	-	-	-	-	3	-	3	5	1	-
AG	-	-	-	-	-	-	-	-	2	4	2	-	-	1
S	-	-	-	-	-	-	-	-	1	-	-	-	-	-

AOI	A	B	F	H	J	T	D	E	G	C	AG	S	AH
A	-	33.33	33.33	33.33	-	-	-	-	-	-	-	-	-
B	-	-	-	-	100.0	-	-	-	-	-	-	-	-
J	-	-	-	-	-	100.0	-	-	-	-	-	-	-
T	100.0	-	-	-	-	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	100.0	-	-	-	-	-	-
D	-	-	-	66.67	-	-	-	33.33	-	-	-	-	-
H	-	-	-	-	-	-	-	88.89	11.11	-	-	-	-
E	7.14	-	-	42.86	-	-	14.29	-	21.43	14.29	-	-	-
C	-	-	-	-	-	-	-	-	42.86	-	57.14	-	-
G	-	-	-	-	-	-	-	25.0	-	25.0	41.67	8.33	-
AG	-	-	-	-	-	-	-	22.22	44.44	22.22	-	-	11.11
S	-	-	-	-	-	-	-	-	100.0	-	-	-	-

Fig. 22. Transition matrix for CWE-79 for one student.

considerable number of arrows in a state diagram and thus become unreadable. This can be shown in Fig. 23. One solution is to reduce the state diagram lines to the major grouping of AOIs. For example, this could be used to match up with the column headers in our swimlane diagram. As shown in the Fig. 24, this improves the readability for the same secure coder that is shown in Fig. 23. It is our determination that state diagrams have limitations as the number of AOIs increase as well as if participants have several transitions between stimuli content or AOIs.

6.7. *Visualization of our eye pupil diameter adjustments*

Prior research has discovered correlations between the changes in our pupil diameter and mental effort or cognitive load when working a problem. Researchers have observed that an increase in pupil dilation occurs when the task complexity increases [70,109–112]. Researchers have examined changes in the pupil diameter for air traffic controllers [113] and participants contributing in a driving simulator [69]. However, to our knowledge, research in pupil changes in the area of secure coding hands-on training modules has not been investigated. Our research examined various techniques to visualize changes in the pupil diameter size for when our secure coders worked our secure coding exercises.

Our custom Python program parsed the eye diameter that was recorded by our selected eye tracking technology at a sampling rate of 60 Hz. This allowed for the generation of various visualization plots of the pupil data. Our initial attempt involved examining the changes in pupil diameter for each AOI. This allowed us to generate a boxplot per each AOI. An example of one secure coder for their left pupil and right pupil is shown in Fig. 25. A second example for another secure coder is shown in Fig. 26. The AOI visited is shown on the X-axis with the pupil diameter shown on the Y-axis. The number of fixations is then overlayed as a second Y-axis.

In our analysis of several secure coders, it was observed that some participants had higher pupil diameters sizes in some AOIs than compared to other AOIs that the secure coder visited. For example, in Fig. 25, is for a secure coder left eye in the top-half and the bottom-half is for the same secure coder right eye. Fig. 25 indicates that several of the peaks in pupil size occurred in the Instruction AOI, the source code function for Getting User Notifications AOI and in the source code function for Sending Notifications to Another User AOI. In Fig. 26, a second secure coder also had peaks in their pupil diameter for the same AOIs as well as when the second secure coder used the Google Search

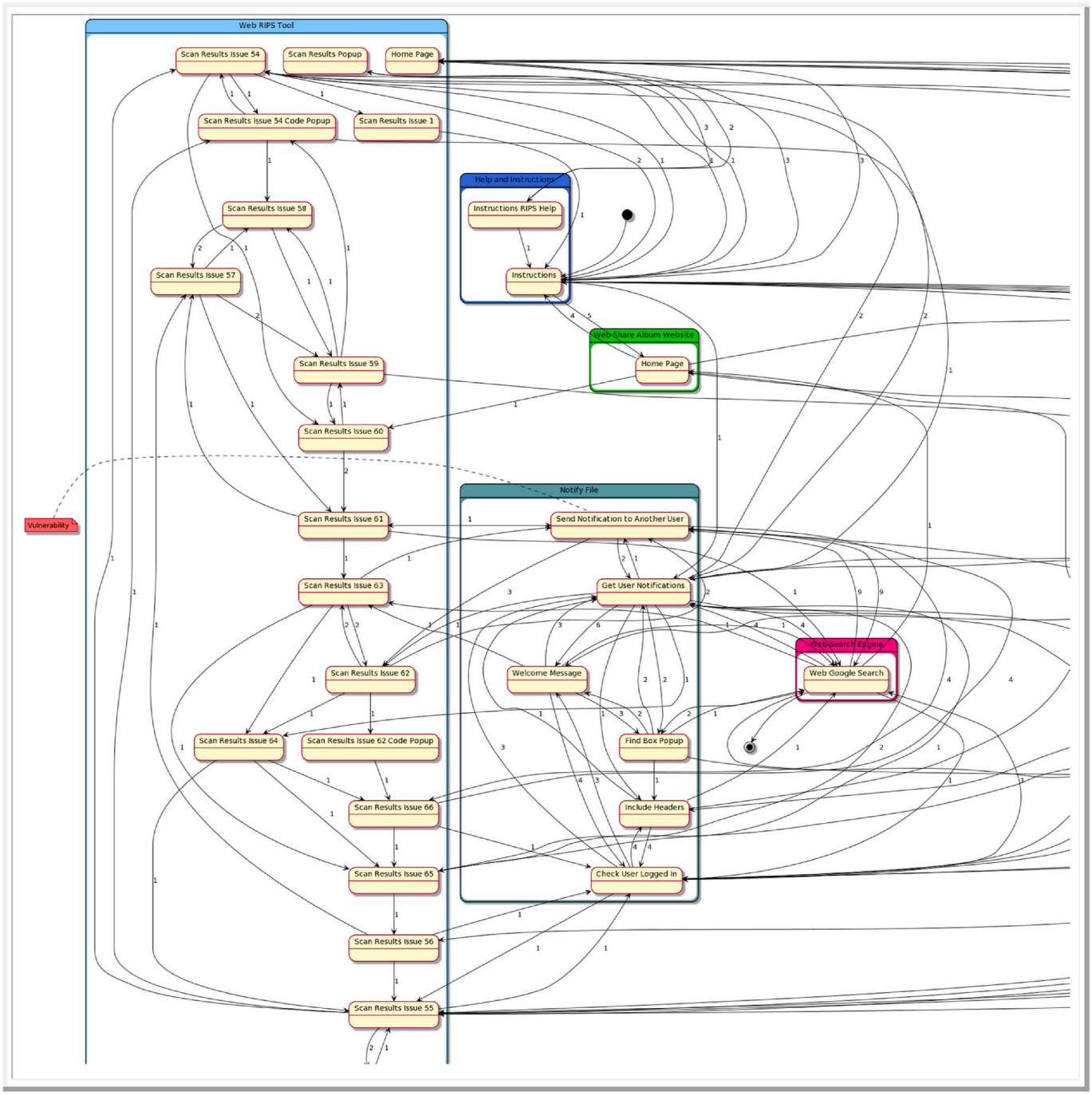


Fig. 23. Complex PlantUML state representation.

Engine. However, a concern with this visualization approach is that when a secure coder enters a AOI that has many fixations, then this technique could potentially miss certain peaks and dips in the pupil diameter since this approach only resets when the secure coders transitions from one AOI to another AOI. Therefore, our analysis was expanded to examine the pupil diameter at a constant rate of time regardless of changing to a different AOI or remaining in the same AOI.

The examination of the pupil diameter for secure coding is a challenging problem due to the potential to miss sudden adjustments in the pupil diameter when viewing eye tracking stimuli that spans multiple AOIs. Our research involved visual techniques to examine the changes in pupil diameters. Future research is needed to determine the correlation between the changes in pupil diameter and participants solving secure coding problems. Our focus in this manuscript is focused on what type of visualizations techniques could potentially benefit the analysis of eye pupil changes over non-linear dynamic stimuli content.

Another technique for examining the left and right pupil data could potentially provide insight when participants do not make frequent transitions between AOIs. This method focused on analyzing the eye tracking data at specific intervals of time. Our algorithm utilized a timestamp in milliseconds that sampled the left and right pupil diameters at a fixed time interval. There is a minimal time spacing that must occur before a new pupil diameter sample is taken from the full recorded dataset. If the sample time window occurs in an AOI that is not of interest, then the data sample is not taken until the participant enters into any AOI that is of interest. Since we are analyzing specific AOIs that

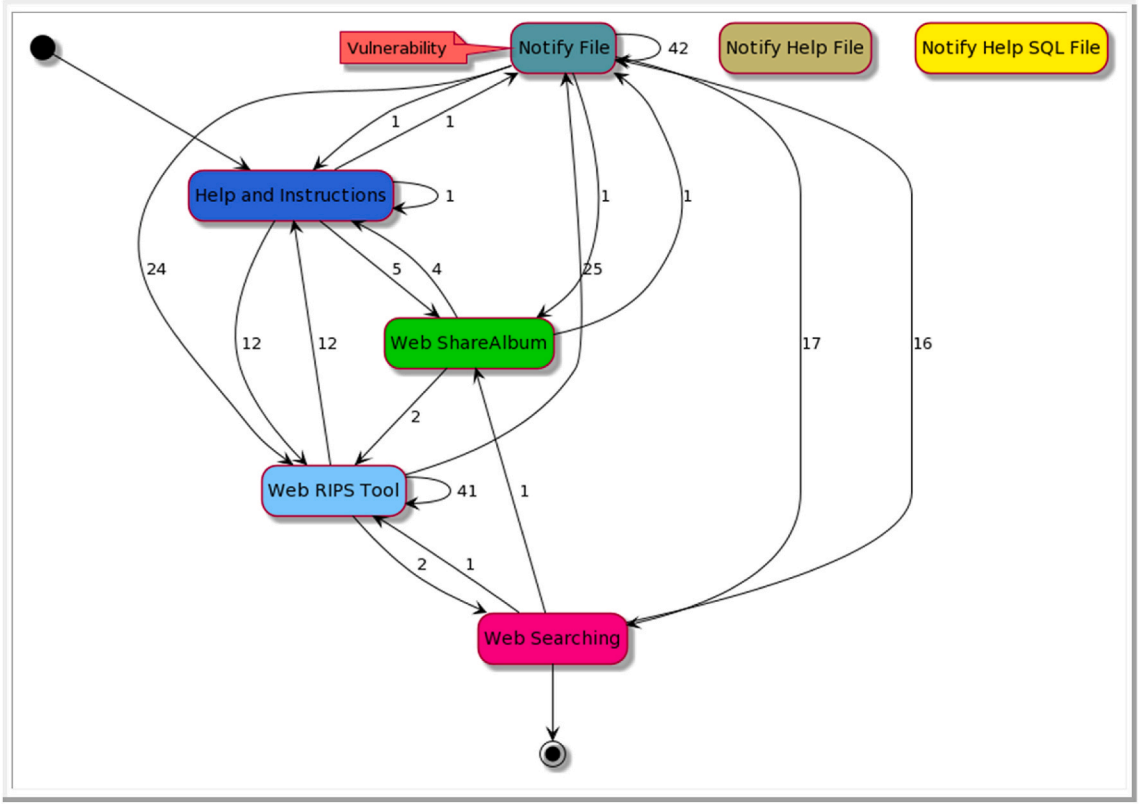


Fig. 24. Simple PlantUML state representation.

are based upon specific CWEs, our pupil sampling algorithm had to manage the occurrences of secure coders viewing AOIs of other CWEs or unrelated stimuli content. This is to eliminate the potential of unwanted AOIs being used in the data sampling for pupil data. Therefore, the smallest interval in time when pupil data is sampled from the full dataset is at least the size of the selected time step size parameter. Our experiments have examined time step parameter sizes of 250 ms, five hundred milliseconds, 750 ms, and one thousand milliseconds. In Fig. 27, we present the left and right pupil measurements for a software developer at an interval of 250 ms for CWE-79. In Fig. 28, we present another software developer at the same interval for CWE-79.

Another approach examined the changes in secure coders pupil diameter by utilizing a Simple Moving Average (SMA) and calculating the mean of the pupil diameter for specific time-periods. In our research, we examined small time-periods such as counting as little as two fixations to as high as counting thirty fixations for the SMA algorithm. An example of a software developer left pupil diameter over a linear timeline with the SMA algorithm and a count of five points is shown in Fig. 29. This specific developer had a steady increase in their left pupil when they examined the software programming function associated with deleting an album from a database. For this specific hands-on programming exercise, a software vulnerability existed in the Delete Album software function. The software vulnerability would allow a security attacker to delete an album even if the security attacker did not create the album. This software developer had multiple reductions in their pupil size after the steady increase levels off. This reduction in the pupil diameter size may indicate a reduction is the cognitive load and processing of the secure coder as a potential indication that the developer believes they have correctly found the software vulnerability [109].

7. Validation of participants responses

Our manual code analysis learning module as well as our learning module that aids by using the static code analysis tool (RIPS) had three overall phrases. The first phrase was the presentation of critical cybersecurity flaws that can exist in source code as determined by the CWE repository and NVD database [4]. This included a brief explanation of how the vulnerability could exist in our photo and video sharing web-based application. The second phrase had either multiple-choice programming options or true/false questions that each participant was required to answer after reviewing static source code presented in Tobii Studio. The multiple-choice questions allowed each participant to select what they believed would best mitigate a security vulnerability. The final phrase had full source code files presented in the Notepad++ text editor that each participant was expected to directly modify the source code. Each participant was asked the same security flaw question; however, no participant was ever allowed to see the response from any other participant.

The verification of each participant response depended upon the type of stimuli that was utilized. For the multiple-choice and true/false questions, the verification of each participant response was immediate and decisive using Tobii Studio questionnaire stimuli. The programming problems presented; however, were manually analyzed afterwards by code review. The programming problems were more complex for participants than the true/false or multiple-choice questions. Our review process of the participants programming problems involved examining the source code that each participant modified and decide if new source code had been written, if pseudocode was provided by the participant, or if just a comment was left in the source code explaining the location and mitigation strategy that should be utilized. Each response was determined to be correct in fully mitigating the vulnerability, partially mitigating the vulnerability or unsuccessful in determining a mitigation technique for the vulnerability. The programming problems were designed to determine if a participant could determine the correct location in the source code files that a vulnerability existed and then, if possible, could develop a mitigation plan to reduce or eliminate the vulnerability. Since not all participants had domain knowledge in the

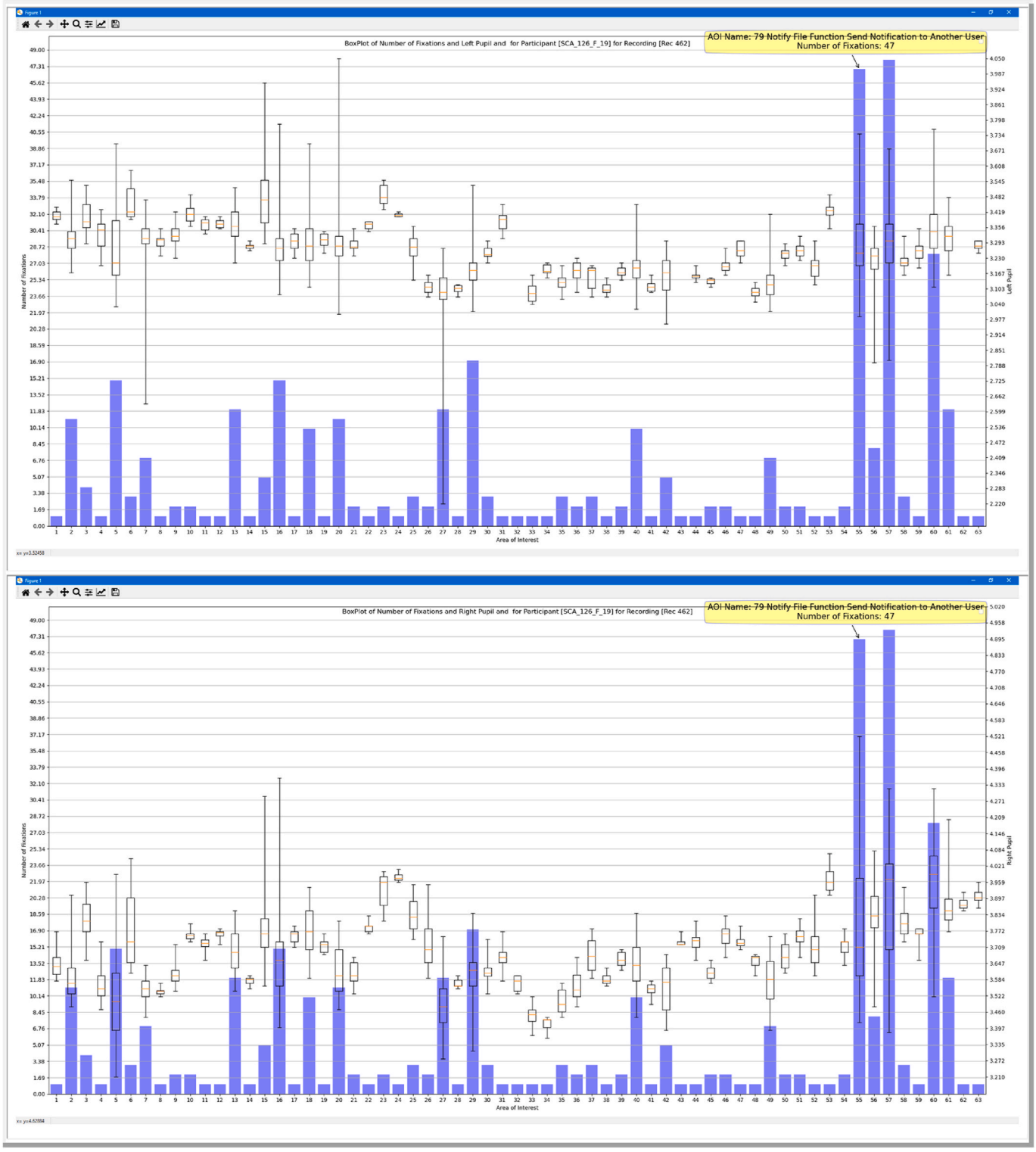


Fig. 25. Boxplot of eye pupil for one student with number of fixations.

HTML and PHP programming languages, participants were allowed the option to provide just pseudocode or a comment if they could not develop the actual source code to mitigate the vulnerability. Each participant was encouraged to attempt to develop the source code if possible and only as an alternative option write the source code or comments. The benefit of at least allowing participants to provide some feedback is the ability to examine if the participant found the location in the source code that the software security flaws exist. This is useful even if they can't develop the source code to resolve the issue. No participant was ever allowed to see the answers of any other participant.

Fig. 30 presents the findings from our first learning module that consisted of a manual code review of the photo and video sharing web-based application. The cybersecurity coding flaw associated with CWE-311 and CWE-434 was presented as multiple-choice questions. Overall, the participants performed well in picking the correct choice that resolved the security flaw. For the coding flaw associated with CWE-

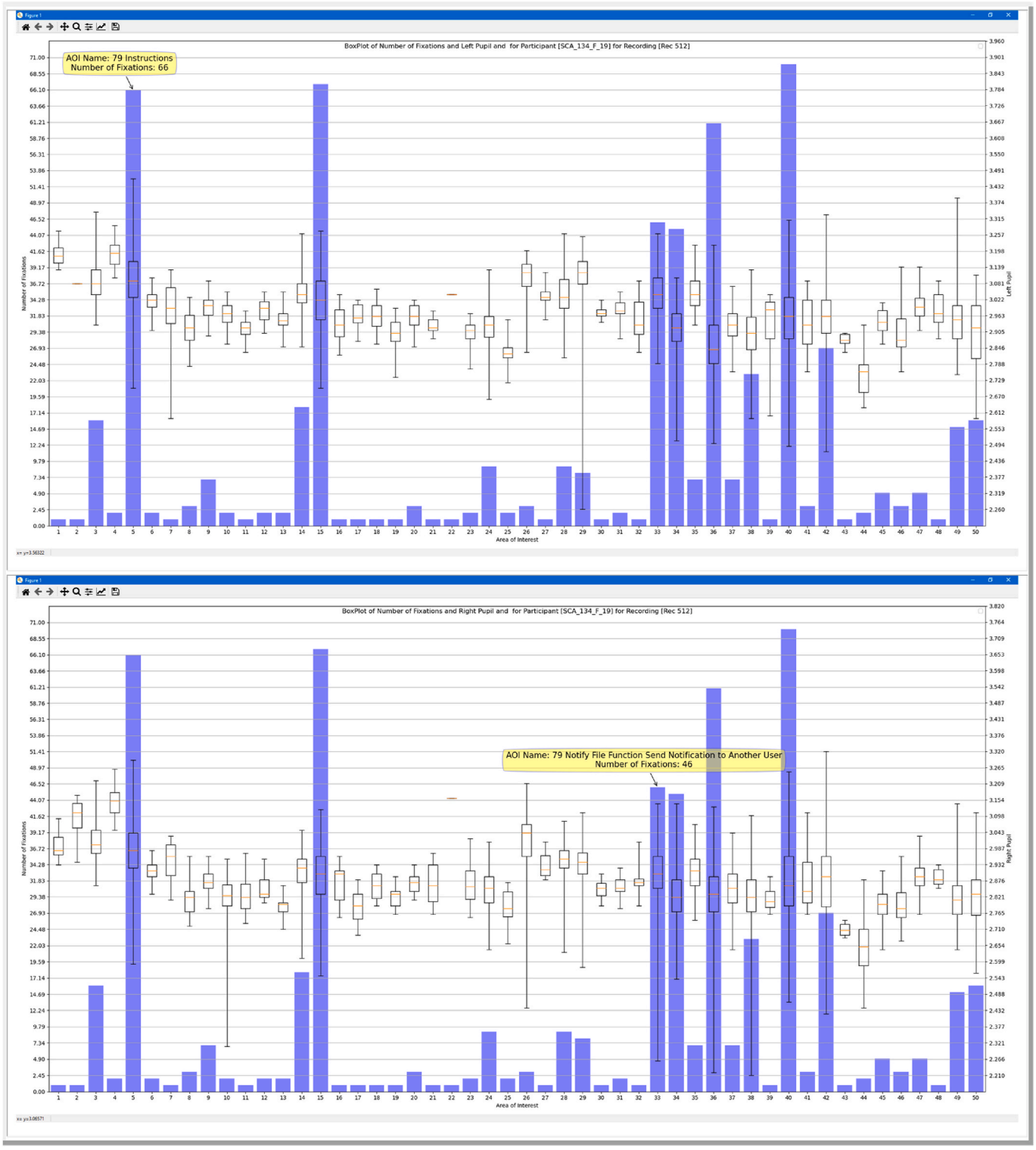


Fig. 26. Boxplot of eye pupil for one student with number of fixations.

862, the participants overall were able to correctly provide source code that mitigated the vulnerability. For the CWE-22 vulnerability, only 14 out of the 29 students correctly determined the flaw location and were able to mitigate the coding flaw.

The second learning module results are presented in Fig. 31. In this learning module, we presented two true/false questions associated with CWE-443 and CWE-73. For these two questions, each participant was asked to answer: 1) True, if the vulnerability existed in the source code as identified by the RIPS static code analysis tool; or 2) False, if the vulnerability did not exist in the source that was identified by the RIPS tool as a potential security flaw. For the coding problem in the second learning module, we focused on CWE-79. Overall, most participants got this programming problem partially correct; very few completely mitigated the vulnerability or completely incorrectly answered the problem.

Our verification of what solutions each participant provided was not directly performed using eye tracking technologies for every question

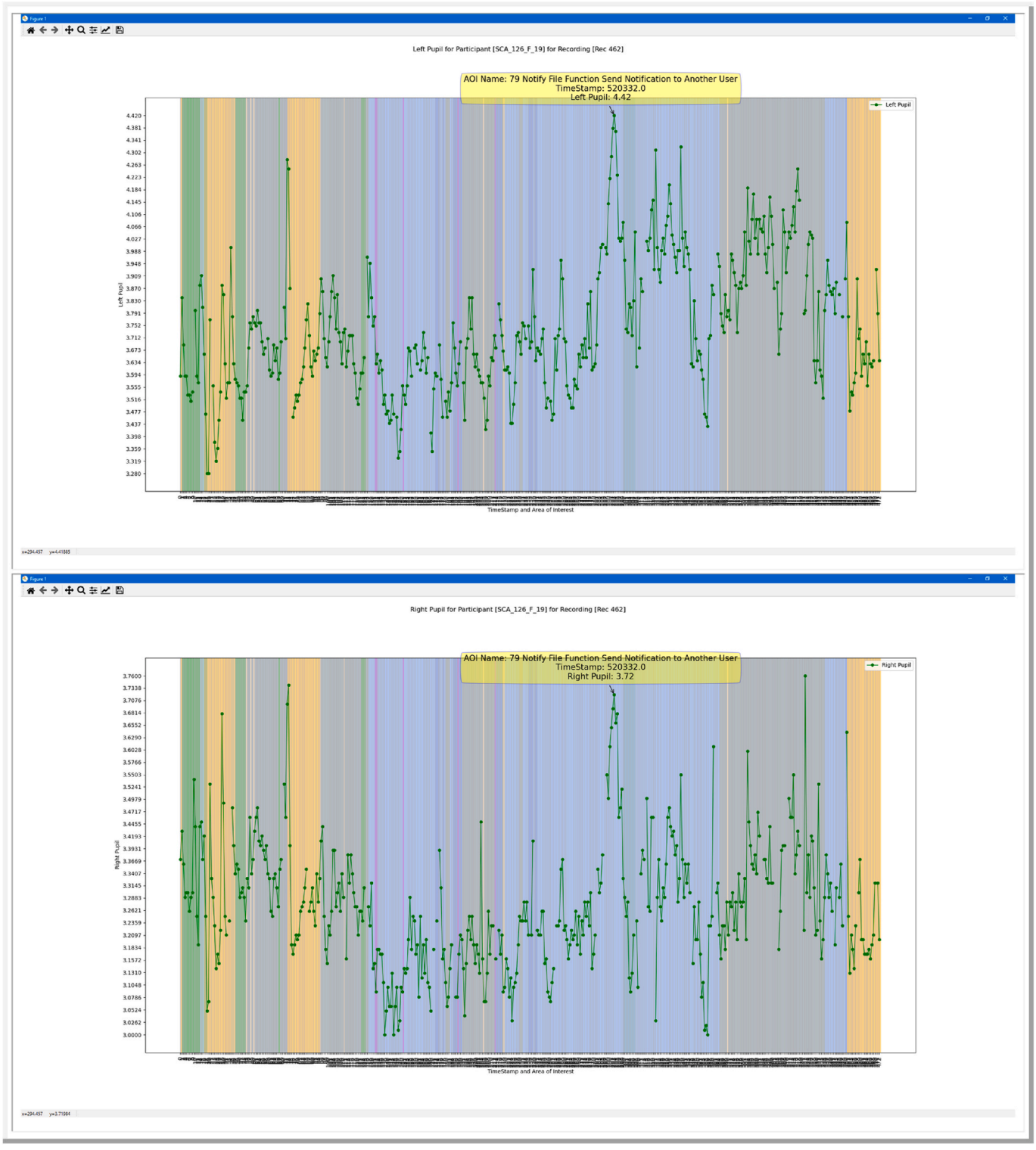


Fig. 27. Eye pupil data for developers at 250 millisecond interval.

that we presented in the eye tracking study. Some of the questions, like multiple choice and true/false, was presented to each participant in the Tobii Studio questionnaire stimuli and the results were collected by Tobii Studio. However, the programming problems required manual code analysis review by researchers and expert coders and documenting the participant response.

8. Conclusions and future work

Our research is focused on examining existing visualization methods and crafting new visualization diagramming techniques that allow for examining the behavior of participants for secure coding projects and objectively visualize secure coders' behavior and patterns. This is critical in order to understand how secure coders approach discovering security weaknesses, their usage of security analysis tools and their

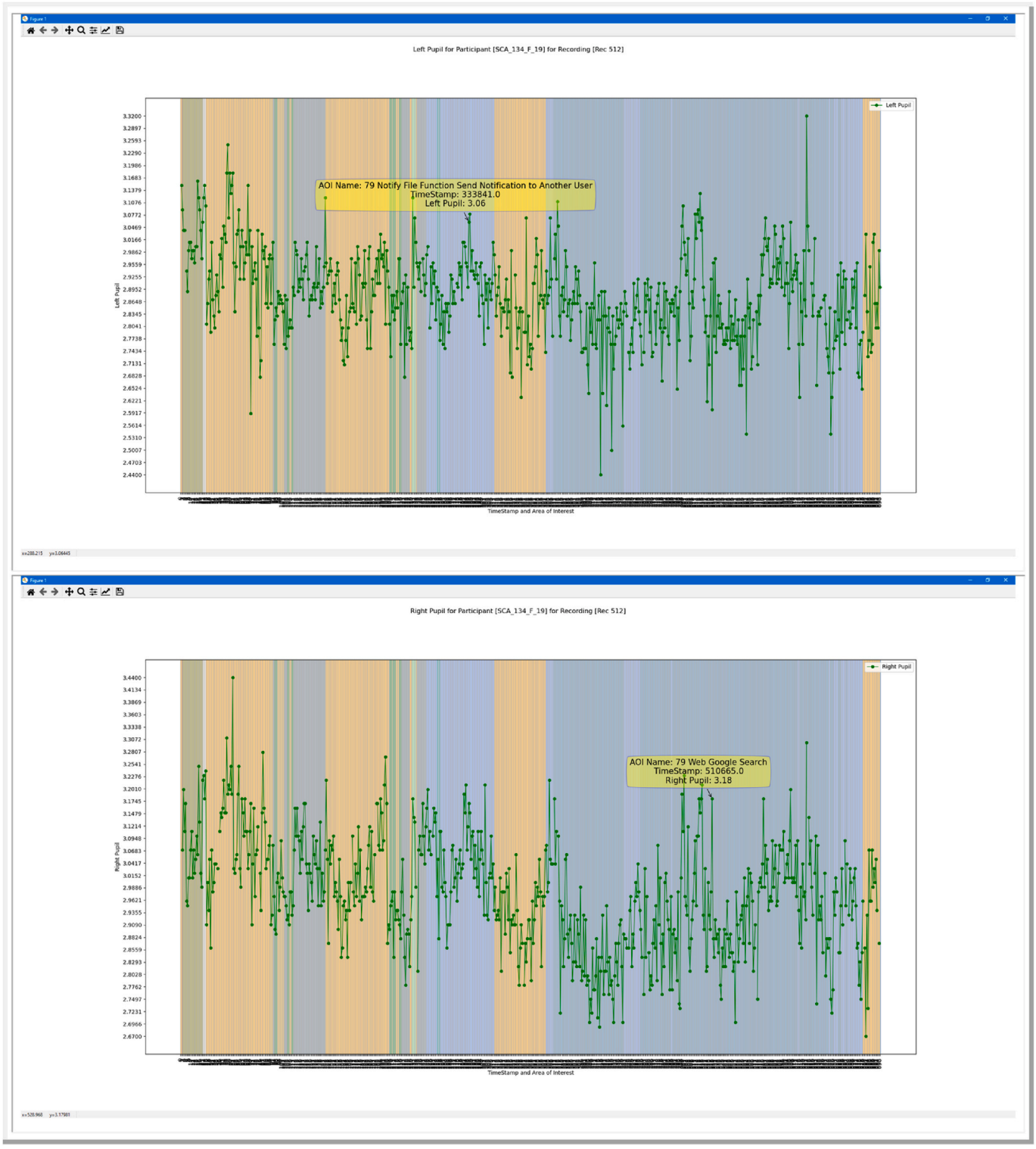


Fig. 28. Eye pupil data for developers at 250 millisecond interval.

approaches to security mitigation techniques.

Our research study utilized heat maps, scan paths, and clusters which are commonly utilized techniques to investigate eye gazes of participants. Then, we explored visual plotting of transitions and AOIs to compare the transition patterns between AOIs. Our approach allowed us to study the reading patterns of participants as they reviewed source code and utilized the security analysis tools. These additional visualization methods allowed another viewpoint to overcome some of the limitations of the simpler techniques to manage several AOIs as well as the transitions between several AOIs. This allowed for exploring the dynamic nature of scrolling and jumping in and out of source code that typically limits the ability to use common eye tracking visualization methods. Our techniques allowed for investigating participants eye gaze at the application level, source code level and then visualize their techniques and strategies that was used to learn and work secure coding exercises.

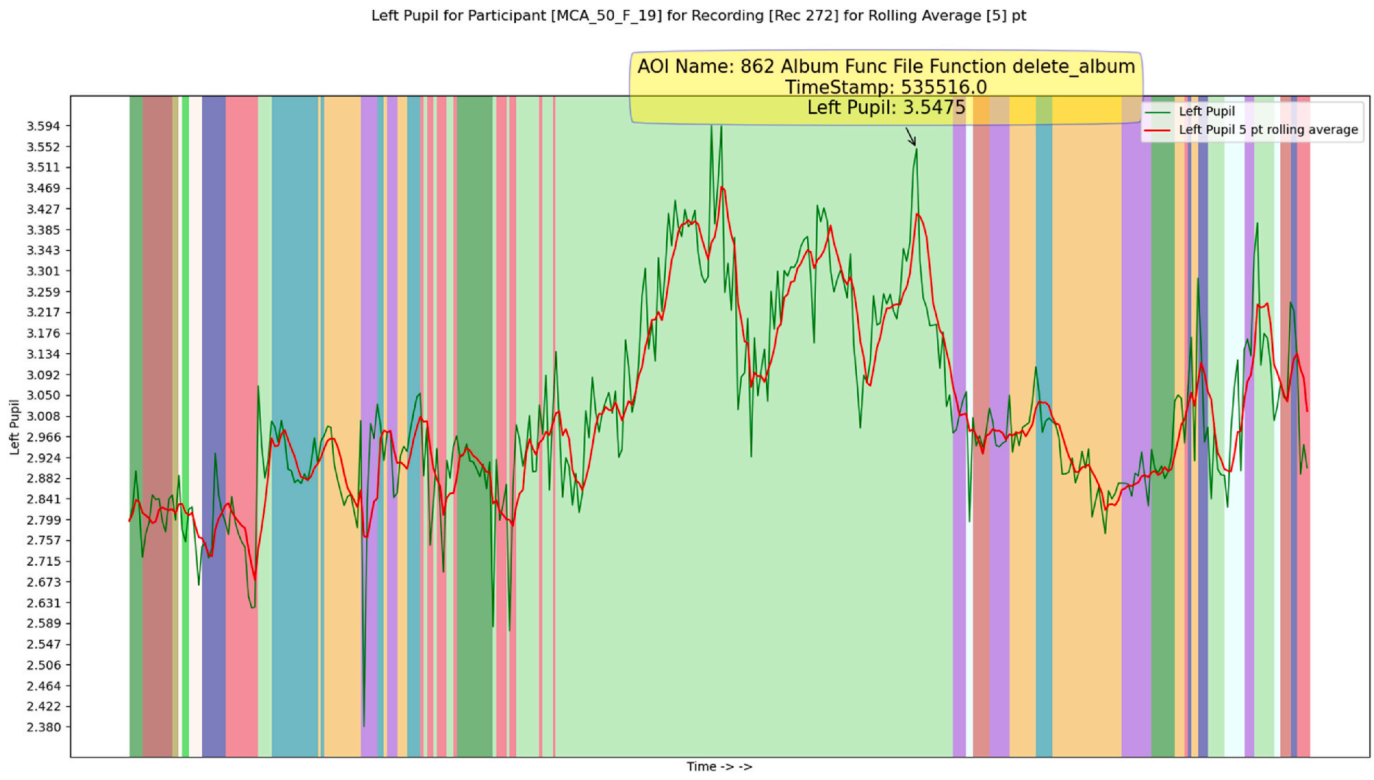


Fig. 29. Simple moving average mean of pupil diameter for a developer.

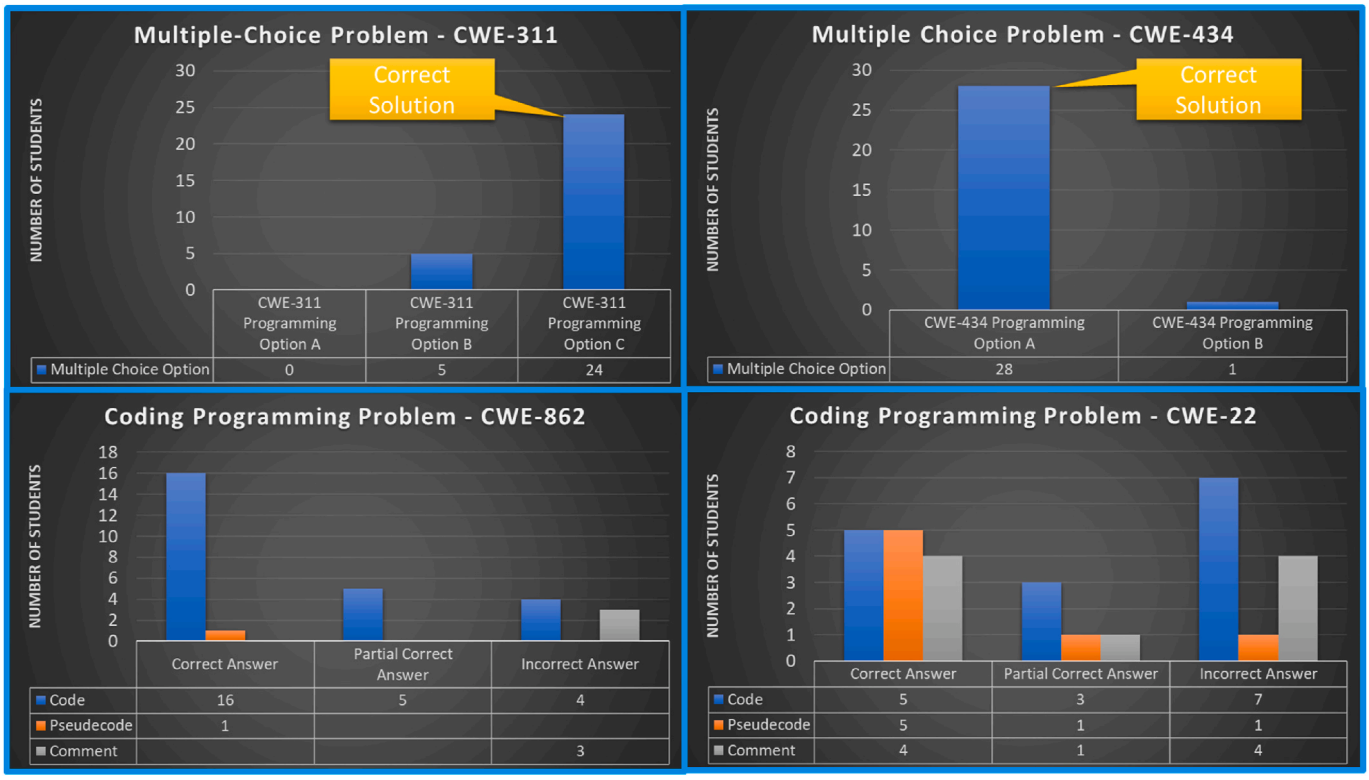


Fig. 30. Manual code analysis learning module participant questionnaire results.

An Area of Interest (AOI) to Fixation Duration diagram allowed us to view the specific AOI that had higher or lower duration of fixations before transitioning to a different AOI. This provided benefits over the simpler fixation duration matrix that simply provided a sum of fixation duration per each AOI. This permitted us to determine over a time period when the participants had higher or lower fixation duration in an AOI. The combination of fixation duration, AOIs and a timeline provided significantly more insight of their behavior versus viewing this

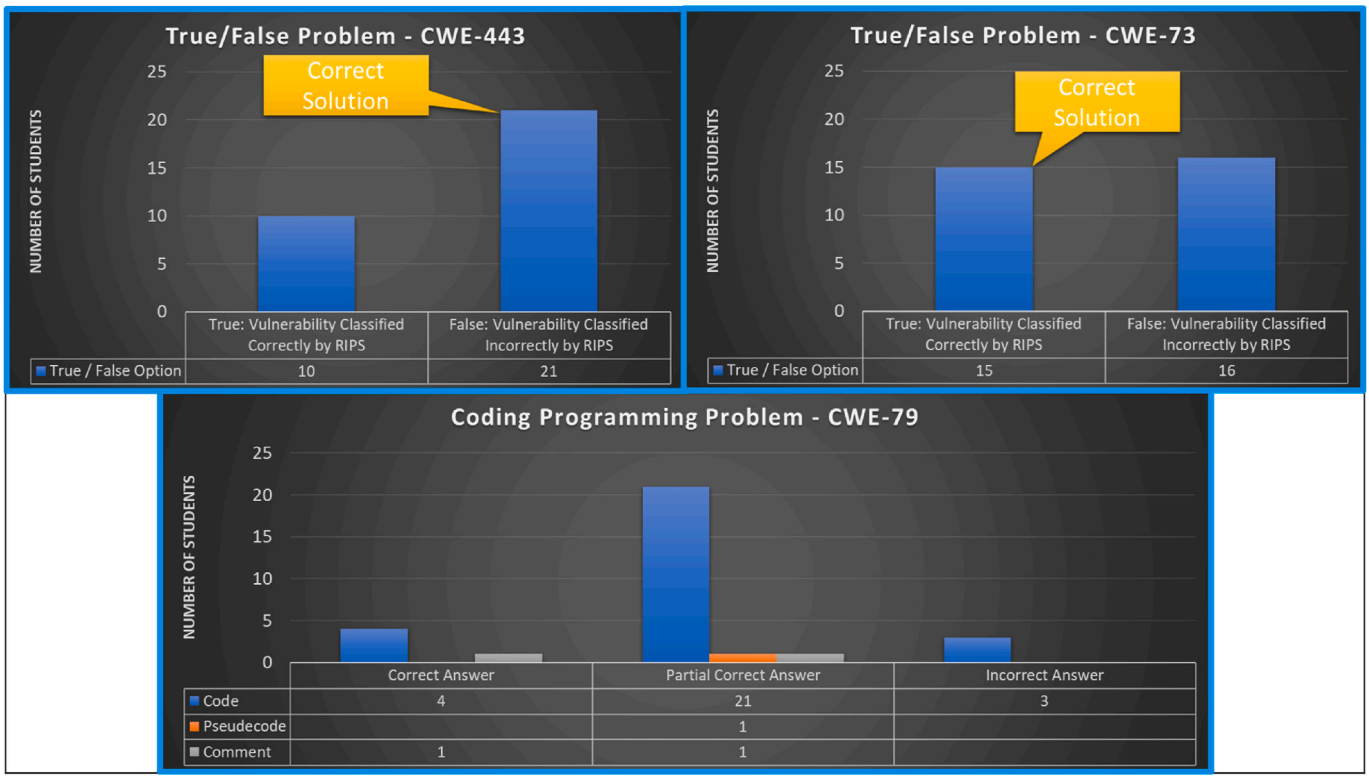


Fig. 31. Static code analysis tool learning module participant questionnaire results.

information separately. This diagramming technique provided insight into when in time and which AOIs each secure coder had higher number of fixations.

We determined that applying a grouping to several of the AOIs could be beneficial to visually examine when participants switch between major applications or subtask such as reading instructions, reading source code, using static code analysis tools, and searching the web for assistance. We created a swim lane representation of the major components using PlantUML. This allowed us to automate the diagram drawing process using Python scripting. We also created a state representation of the major components and created another diagramming method using PlantUML as well. This technique is designed to give a high-level overview of the transitions between major components.

Additionally, we examined the left and right pupil data that was recorded during our eye recording sessions. Following a similar approach to visualize the pupil data, our work focused on creating custom script to examine the minimum, maximum and average pupil diameter changes over a timeline and for our AOIs. We also created the ability to sample the eye pupil size at specific intervals in time to keep from losing data points when participants do not transition out of an AOI for an extended period of time. To our knowledge, the literature is lacking in research that examines the pupil changes in regard to secure coding. Our future research is to ascertain if insight can be determined as to when participants discover a vulnerability by using their pupil diameter size.

Our research indicates that visualization methods can allow for the ability to study eye gaze for participants conducting secure coding activities. The visualization of the data can allow for gaining insight into participants behavior and patterns among participants. The current literature has limited visualization methods for examining secure coders eye gazes while they learn and work secure coding problems. Questionnaires and review sessions cannot impartially provide adequate feedback on software engineers learning techniques and their approaches to problem solving. Eye tracking technologies offer the ability to objectively observe and measure software developers learning patterns and techniques in how they approach secure coding and software development.

Our current research work is focused on gaining insight by examining the patterns in the behaviors of software developers that participated in our research study. Specifically, our interest is in examining the behavior when participants utilize multiple applications and multiple stimuli content in secure coding activities. Understanding of user behaviors and patterns when they utilize multiple applications and when users control the ordering of stimuli events for eye tracking has not been extensively examined. Our belief is that by visualizing the various eye gaze transitions between multiple applications, then we can gain insight into the behavioral techniques and gaze patterns that secure coders utilized to solve problems. This research involves using the visualizations techniques discussed in this paper to specify specific reading patterns that participants use when reading learning content as well as source code to mitigate security flaws. Additionally, we desire to examine how the usage of coding tools facilitate or hinder software developers' ability to work secure coding problems.

Authors' contributions

Daniel Kyle Davis: Conceptualization, Methodology, Data Curation, Formal Analysis, Investigation, Software, Visualization, Writing – Original Draft. Feng Zhu: Conceptualization, Methodology, Investigation, Formal Analysis, Writing – Review & Editing.

Funding source

This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Studies in humans

Institutional Review Board Approval for Eye Tracking Research Study.

The Principal Investigator created a request to conduct an eye tracking research study and submitted it to their university Institutional Review Board. The research study requested authorization to conduct an eye tracking research study with students from the university. The Institutional Review Board at the university approved the research study. Written consent from every student was mandatory before any eye tracking sessions commenced. The raw eye gaze data for every student is confidential and has not been publicly released.

Availability of data and material

The eye gaze datasets for our visual analysis are currently not publicly available for accessing. This is due to the raw eye gaze datasets containing private student information that is protected by our Institutional Review Board requirements. In order to publicly publish the raw eye gaze data, we would require the consent from every student before releasing their individual raw eye gaze datasets. Currently, we do not have authorization to release participants' raw eye gaze data that is confidential.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

None.

References

[1] Gluck M, Mercado E, Myers C. Learning and memory from brain to behavior. Fourth. Worth Publishers; 2020.
[2] Fan M, Shi S, Truong KN. Practices and challenges of using think-aloud protocols in industry: an international survey. J Usability Stud 2020;15:85–102.
[3] Schiela R, Sherman M. From secure coding to secure software. Carnegie Mellon University; 2016.
[4] Stock A Van Der, Lowery D, Rook D, Cruz D, Keary E, Williams J, et al. OWASP code review guide. 2008., Version 1.1.
[5] OWASP Foundation. OWASP vulnerability management model. 2020.
[6] The MITRE Corporation. CVE—common vulnerabilities and exposures—the standard for information security vulnerability names. 2013. http://cve.mitre.org/index.html. [Accessed 5 May 2019].
[7] The MITRE Corporation. Common Attack pattern enumeration and classification (CAPEC™). 2016. http://capec.mitre.org/data/. [Accessed 4 September 2019].
[8] The MITRE Corporation. Common weakness enumeration. 2019. https://cwe.mitre.org/about/index.html. [Accessed 9 August 2019].
[9] Keary E, Conklin L, Robinson G. OWASP code review guide, V2.0. 2017.
[10] Nichols LE, Kabay ME, Braithwaite T. Writing secure code. second ed. Washington: Redmond; 2003. https://doi.org/10.1002/9781118851678.ch38.
[11] Long F, Mohindra D, Seacord R, Sutherland D, Svoboda D. The CERT oracle secure coding standard for java. Addison-Wesley Professional; 2011.
[12] Long F, Seacord R. Java coding guidelines: 75 recommendations for reliable and secure programs: 75 recommendations for reliable and secure programs (SEI series in software engineering). Software Engineering Institute; 2014.
[13] Seacord R. The CERT C secure coding standard. first ed. Boston, MA: Addison-Wesley Professional; 2009.
[14] Seacord R. Secure coding in C and C++. second ed. Addison-Wesley Professional; 2006.
[15] OWASP. OWASP ZAP. n.d. https://owasp.org/www-project-zap/. [Accessed 10 September 2019].
[16] Veracode. Veracode static analysis. n.d. https://www.veracode.com/products/binary-static-analysis-sast. [Accessed 4 August 2019].
[17] Software Engineering Institute (SEI) at Carnegie Mellon University. Curricula: software assurance materials and artifacts. 2014.
[18] OWASP Foundation. OWASP secure coding practices – quick reference guide. 2010.
[19] Taylor B, Kaza S. Security injections: modules to help students remember, understand, and apply secure coding techniques. In: Proceedings of the 16th annual joint conference on innovation and technology in computer science education. ACM; 2011. p. 3–7.
[20] Du W, Teng Z, Wang R. SEED: a suite of instructional laboratories for computer SEcurity EDucation. In: Proceedings of the 38th SIGCSE technical symposium on computer science education. ACM; 2007. p. 486–90.
[21] Software Engineering Institute (SEI) at Carnegie Mellon University. Software assurance curricula. https://www.sei.cmu.edu/education-outreach/curricula/software-assurance/index.cfm. [Accessed 6 June 2019].
[22] CERIAS PU. Security awareness, education, and training. 2021. https://www.cerias.purdue.edu/research/projects/home/area/11/security_awareness_education_and_training. [Accessed 14 April 2019].
[23] The Security Injections@Towson. Cyber4All security injections. http://cisserv1.towson.edu/~cyber4all/index.php/security-injections_home/. [Accessed 12 August 2020].
[24] Georgia Institute of Technology. Secure software development. 2021. https://pe.gatech.edu/courses/secure-software-development. [Accessed 5 April 2019].
[25] Du W. SEEDLABS 2.0. Syracuse University; 2020. https://seedsecuritylabs.org/Labs_20.04/Software/. [Accessed 6 June 2019].
[26] OWASP Open Web Application Security Project. OWASP secure coding Dojo. https://owasp.org/www-project-secure-coding-dojo/. [Accessed 5 May 2019].
[27] SAFECode. About SAFECode training. https://safecode.org/training/about-this-program/. [Accessed 15 August 2020].
[28] Veracode. Veracode security labs community edition. 2021. https://www.veracode.com/products/security-labs-community-edition. [Accessed 24 April 2019].
[29] HackEDU. Hands-on secure development training that's easy to set up and deploy. 2021. https://www.hackedu.com/secure-development-training. [Accessed 7 July 2020].
[30] Avatao. Start with secure coding & deliver high quality products. 2021. https://avatao.com/our-pricing/. [Accessed 5 April 2019].
[31] Secure Code Warrior. Secure code warrior. 2021. https://www.securecodewarrior.com/company/about-us. [Accessed 5 July 2019].
[32] The MITRE Corporation. Overview - what is CWE? The MITRE Corporation n.d. https://cwe.mitre.org/about/index.html (accessed June 5, 2019).
[33] The MITRE Corporation. CWE top 25 most dangerous software weaknesses. The MITRE Corporation n; 2020. d. https://cwe.mitre.org/top25/archive/2020/2020_cwe_top25.html. [Accessed 1 August 2020].
[34] National Institute of Standards and Technology (NIST). National vulnerability Database. National Institute of Standards and Technology n.d. https://nvd.nist.gov/vuln/categories (Accessed 10 June 2019).
[35] Schilling K, Applegate R. Best methods for evaluating educational impact: a comparison of the efficacy of commonly used measures of library instruction. J Med Libr Assoc 2012;100:258–69. https://doi.org/10.3163/1536-5050.100.4.007.
[36] Fox MA, Norman H, NRC. Evaluating and improving undergraduate teaching in science, mathematics, engineering, and technology. Washington: National Academy Press; 2003.
[37] Friedman N, Kaiser P, Trattler W. Review of ophthalmology. Elsevier; 2017.
[38] Zhao L, Wang Z, Zhang G, Qi Y, Wang X. Eye state recognition based on deep integrated neural network and transfer learning. Multimed Tool Appl 2018;77:19415–38. https://doi.org/10.1007/s11042-017-5380-8.
[39] Prabhakar G, Ramakrishnan A, Madan M, Murthy LRD, Sharma VK, Deshmukh S, et al. Interactive gaze and finger controlled HUD for cars. J Multimodal User Interfaces 2020;14:101–21. https://doi.org/10.1007/s12193-019-00316-9.
[40] IMotions. Top 8 eye tracking applications in research. IMotions; 2015 (accessed March 5, 2019), https://imotions.com/blog/top-8-applications-eye-tracking-research/.
[41] Le Louedec J, Guntz T, Crowley JL, Vaufreydaz D. Deep learning investigation for chess player attention prediction using eye-tracking and game data. In: Eye tracking research and applications symposium (ETRA); 2019. p. 1–15. https://doi.org/10.1145/3314111.3319827.
[42] Dill ET, Young SD, Facility CM, Aerospace S. Analysis of eye-tracking data with regards to the complexity of flight deck information automation and management – inattentional blindness, system state awareness, and EFB usage. 2015. Hampton.
[43] Skvarekova I, Skultety F. Objective measurement of pilot's attention using eye track technology during IFR flights. Transport Res Procedia 2019;40:1555–62. https://doi.org/10.1016/j.trpro.2019.07.215.
[44] Glaholt MG. Eye tracking in the cockpit: a review of the relationships between eye movements and the aviator's cognitive state. Drdc-Rddc-2014-R153; 2014. p. 1–58.
[45] De Smet B, Lempereur L, Sharafi Z, Guéhéneuc YG, Antoniol G, Habra N. Taupe: visualizing and analyzing eye-tracking data. Sci Comput Program 2014;79:260–78. https://doi.org/10.1016/j.scico.2012.01.004.
[46] Holmqvist K, Nyström M, Andersson R, Dewhurst R, Jarodzka H, Weijer J van de. Eye tracking A comprehensive guide to methods and measures. 2013. https://doi.org/10.1017/CBO9781107415324.004.
[47] Bednarik R, Tukiainen M. An eye-tracking methodology for characterizing program comprehension processes. In: Eye tracking research and applications symposium (ETRA) 2005; 2005. p. 125–32. https://doi.org/10.1145/1117309.1117356.
[48] Pro Tobii. Tobii Studio user's manual. 2016. https://doi.org/10.1017/CBO9781107415324.004., Version 3.4.5.
[49] Duchowski A. Eye tracking methodology: theory and practice. second ed. Springer-Verlag London; 2007. https://doi.org/10.1007/978-1-84628-609-4.
[50] Blascheck T, Sharif B. Visually analyzing eye movements on natural language texts and source code snippets. In: Eye tracking research and applications symposium (ETRA); 2019. https://doi.org/10.1145/3314111.3319917.
[51] Kurzhals K, Weiskopf D. AOI transition trees. Graph Interface June, 2015;2015:41–8.

[52] Warnier T, Rikmenspoel O, Puente T, Peeters L, Velde S, Burch M. The future and challenges of eye tracking data visualizations. 2020. https://doi.org/10.13140/RG.2.2.13355.87842.
[53] Francisti J, Balogh Z, Reichel J, Magdin M, Koprda Š, Molnár G. Application experiences using IoT devices in education. Appl Sci (Switzerland) 2020;10:1–14. https://doi.org/10.3390/app10207286.
[54] Blascheck T, Schweizer M, Beck F, Ertl T. Visual comparison of eye movement patterns. Comput Graph Forum 2017;36:87–97. https://doi.org/10.1111/cgf.13170.
[55] Blascheck T, Kurzhals K, Raschke M, Strohmaier S, Weiskopf D, Ertl T. AOI hierarchies for visual exploration of fixation sequences. In: Eye tracking research and applications symposium (ETRA). vol. 14; 2016. p. 111–8. https://doi.org/10.1145/2857491.2857524.
[56] Peterson CS, Saddler J, Blascheck T, Sharif B. Visually analyzing students' gaze on C++ code snippets. In: Proceedings - 2019 IEEE/ACM 6th international workshop on eye movements in programming, EMIP 2019; 2019. p. 18–25. https://doi.org/10.1109/EMIP.2019.00011.
[57] Rayner K. Eye movements in reading and information processing: 20 Years of research. Psychol Bull 1998. https://doi.org/10.1037//0033-2909.124.3.372.
[58] Jbara A, Feitelson DG. How programmers read regular code: a controlled experiment using eye tracking. Empir Software Eng 2017;22:1440–77. https://doi.org/10.1007/s10664-016-9477-x.
[59] Lai ML, Tsai MJ, Yang FY, Hsu CY, Liu TC, Lee SWY, et al. A review of using eye-tracking technology in exploring learning from 2000 to 2012. Educ Res Rev 2013;10:90–115. https://doi.org/10.1016/j.edurev.2013.10.001.
[60] Alemdag E, Cagiltay K. A systematic review of eye tracking research on multimedia learning. Comput Educ 2018;125:413–28. https://doi.org/10.1016/j.compedu.2018.06.023.
[61] Blascheck T, Kurzhals K, Raschke M, Burch M, Weiskopf D, Ertl T. State-of-the-Art of visualization for eye tracking data. In: Eurographics conference on visualization (EuroVis); 2014. p. 1–20. https://doi.org/10.2312/eurovisstar.20141173.
[62] Blascheck T, Kurzhals K, Raschke M, Burch M, Weiskopf D, Ertl T. Visualization of eye tracking data: a taxonomy and survey. Comput Graph Forum 2017;36:260–84. https://doi.org/10.1111/cgf.13079.
[63] Sharafi Z, Soh Z, Guéhéneuc YG. A systematic literature review on the usage of eye-tracking in software engineering. Inf Software Technol 2015;67:79–107. https://doi.org/10.1016/j.infsof.2015.06.008.
[64] Obaidellah U, Al Haek M, Cheng PCH. A survey on the usage of eye-Tracking in computer programming. ACM Comput Surv 2018;51. https://doi.org/10.1145/3145904.
[65] Davis DK, Zhu F. Analysis of software developers' coding behavior: a survey of visualization analysis techniques using eye trackers. Comput Hum Behav Rep 2022;7. https://doi.org/10.1016/j.chbr.2022.100213.
[66] Peitek N, Siegmund J, Parnin C, Apel S, Brechmann A. Toward conjoint analysis of simultaneous eye-tracking and fMRI data for program-comprehension studies. In: Proceedings - EMIP. Eye Movements in Programming 2018; 2018. https://doi.org/10.1145/3216723.3216725.
[67] Sharif B, Peterson CS, Guarnera DT, Bryant CA, Buchanan Z, Zyrianov V, et al. Practical eye tracking with iTrace. In: Proceedings - 2019 IEEE/ACM 6th international workshop on eye movements in programming, EMIP 2019; 2019. p. 41–2. https://doi.org/10.1109/EMIP.2019.00015.
[68] Uwano H, Nakamura M, Monden A, Matsumoto KI. Analyzing individual performance of source code review using reviewers' eye movement. In: Eye tracking research and applications symposium (ETRA) 2005; 2005. p. 133–40. https://doi.org/10.1145/1117309.1117357.
[69] Palinko O, Kun AL, Shyrokov A, Heeman P. Estimating cognitive load using remote eye tracking in a driving simulator. In: Eye tracking research and applications symposium (ETRA); 2010. p. 141–4. https://doi.org/10.1145/1743666.1743701.
[70] Klingner J, Kumar R, Hanrahan P. Measuring the task-evoked pupillary response with a remote eye tracker. In: Eye tracking research and applications symposium (ETRA); 2008. p. 69–72. https://doi.org/10.1145/1344471.1344489. 1.
[71] Abid NJ, Maletic JI, Sharif B. Using developer eye movements to externalize the mental model used in code summarization tasks. In: Eye tracking research and applications symposium (ETRA); 2019. https://doi.org/10.1145/3314111.3319834.
[72] Ahrens M, Schneider K, Busch M. Attention in software maintenance: an eye tracking study. In: Proceedings - 2019 IEEE/ACM 6th international workshop on eye movements in programming, EMIP 2019; 2019. https://doi.org/10.1109/EMIP.2019.00009.
[73] Busjahn T, Bednarik R, Begel A, Crosby M, Paterson JH, Schulte C, et al. Eye movements in code reading: relaxing the linear order. In: IEEE international conference on program comprehension. 2015; August, 2015. p. 255–65. https://doi.org/10.1109/ICPC.2015.36.
[74] Shaffer TR, Wise JL, Walters BM, Müller SC, Falcone M, Sharif B. ITrace: enabling eye tracking on software artifacts within the IDE to support software engineering tasks. In: 2015 10th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ESEC/FSE 2015 - proceedings; 2015. https://doi.org/10.1145/2786805.2803188.
[75] Guarnera DT, Bryant CA, Mishra A, Maletic JI, Sharif B. iTrace: eye tracking infrastructure for development environments. In: Eye tracking research and applications symposium (ETRA); 2018. p. 2015–7. https://doi.org/10.1145/3204493.3208343.
[76] Zyrianov V, Guarnera DT, Peterson CS, Sharif B, Maletic JI. Automated recording and semantics-aware replaying of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks. In: Proceedings - 2020 IEEE international conference on software maintenance and evolution, ICSME 2020; 2020. p. 464–75. https://doi.org/10.1109/ICSME46990.2020.00051.
[77] Ahrens M. Towards automatic capturing of traceability links by combining eye tracking and interaction data. In: Proceedings of the IEEE international conference on requirements engineering 2020; 2020. p. 434–9. https://doi.org/10.1109/RE48521.2020.00064. Augus.
[78] Fakhoury S, Roy D, Pines H, Cleveland T, Peterson CS, Arnaoudova V, et al. Gazel: supporting source code edits in eye-tracking studies. In: Proceedings - international conference on software engineering; 2021. p. 69–72. https://doi.org/10.1109/ICSE-Companion52605.2021.00038.
[79] Stein R, Brennan SE. Another person's eye gaze as a cue in solving programming problems. In: ICMI'04 - sixth international conference on multimodal interfaces. vols. 9–15; 2004. https://doi.org/10.1145/1027933.1027936.
[80] Busjahn T, Schulte C, Busjahn A. Analysis of code reading to gain more insight in program comprehension. In: Proceedings - 11th Koli Calling international conference on computing education research, Koli Calling'11; 2011. https://doi.org/10.1145/2094131.2094133.
[81] Ahrens M, Schneider K. Improving requirements specification use by transferring attention with eye tracking data. Inf Software Technol 2021;131:106483. https://doi.org/10.1016/j.infsof.2020.106483.
[82] Walters B, Shaffer T, Sharif B, Kagdi H. Capturing software traceability links from developers' eye gazes. In: 22nd international conference on program comprehension, ICPC 2014 - proceedings; 2014. p. 201–4. https://doi.org/10.1145/2597008.2597795.
[83] Sharif B, Clark B, Maletic JI. Studying developer gaze to empower software engineering research and practice. In: Proceedings of the ACM SIGSOFT symposium on the foundations of software engineering; 2016. https://doi.org/10.1145/2950290.2983988. 13-18-Nove:940–3.
[84] Walters B, Falcone M, Shibble A, Sharif B. Towards an eye-tracking enabled IDE for software traceability tasks. In: 2013 7th international workshop on traceability in emerging forms of software engineering, TEFSE 2013 - proceedings; 2013. p. 51–4. https://doi.org/10.1109/TEFSE.2013.6620154.
[85] Soh Z, Sharafi Z, Van Den Plas B, Porras GC, Guéhéneuc YG, Antoniol G. Professional status and expertise for UML class diagram comprehension: an empirical study. In: IEEE international conference on program comprehension; 2012. p. 163–72. https://doi.org/10.1109/icpc.2012.6240484.
[86] Sharif B, Kagdi H. On the use of eye tracking in software traceability. In: Proceedings - international conference on software engineering; 2011. p. 67–70. https://doi.org/10.1145/1987856.1987872.
[87] Kagdi H, Yusuf S, Maletic JI. On using eye tracking in empirical assessment of software visualizations. In: Proc - 1st ACM int workshop on empirical assessment of software engineering languages and technologies, WEASELTech 2007, held with the 22nd IEEE/ACM int Conf automated software eng, ASE 2007; 2007. p. 21–2. https://doi.org/10.1145/1353673.1353678.
[88] Sharif B, Shaffer T, Wise J, Maletic JI. Tracking developers' eyes in the IDE. IEEE Softw 2016;33:105–8. https://doi.org/10.1109/MS.2016.84.
[89] Buttner D. The importance of manual secure code review. The MITRE Corporation; 2014. https://www.mitre.org/capabilities/cybersecurity/overview/cybersecurity-blog/the-importance-of-manual-secure-code-review. [Accessed 8 July 2019].
[90] The MITRE Corporation. Secure code review. The MITRE Corporation n.d. https://www.mitre.org/publications/systems-engineering-guide/enterprise-engineering/systems-engineering-for-mission-assurance/secure-code-review (Accessed 8 July, 2019).
[91] Synopsys T, Editorial. Why secure code review matters (and actually saves time!). Synopsys; 2017. https://www.synopsys.com/blogs/software-security/why-secure-code-reviews-matter/. [Accessed 4 April 2019].
[92] Kesäniemi A. Automatic vs. Manual code analysis OWASP. 2009.
[93] Dahse J. RIPS - a static source code analyser for vulnerabilities in PHP scripts. 2018. http://rips-scanner.sourceforge.net/. [Accessed 15 July 2019].
[94] CLASP. CWE-311: missing encryption of sensitive data. The MITRE Corporation; 2006. https://cwe.mitre.org/data/definitions/311.html. [Accessed 5 May 2019].
[95] PLOVER. CWE-434: unrestricted upload of file with dangerous type. The MITRE Corporation; 2006. https://cwe.mitre.org/data/definitions/434.html. [Accessed 5 May 2019].
[96] Pernicious Kingdoms 7. CWE-73: external control of file name or path. The MITRE Corporation; 2006. https://cwe.mitre.org/data/definitions/73.html. [Accessed 5 May 2019].
[97] Community C. CWE-443: DEPRECATED (Duplicate): HTTP response splitting. The MITRE Corporation; 2006 (accessed May 5, 2019), https://cwe.mitre.org/data/definitions/443.html.
[98] PLOVER. CWE-113: improper neutralization of CRLF sequences in HTTP headers ('HTTP response splitting'). The MITRE Corporation; 2006 (accessed May 5, 2019), https://cwe.mitre.org/data/definitions/113.html.
[99] Team CC. CWE-862: missing authorization. The MITRE Corporation; 2011. https://cwe.mitre.org/data/definitions/862.html. [Accessed 5 May 2019].
[100] PLOVER. CWE-22. Improper limitation of a pathname to a restricted directory ('Path traversal'). The MITRE Corporation; 2006. https://cwe.mitre.org/data/definitions/22.html. [Accessed 5 May 2019].
[101] PLOVER. CWE-79: improper neutralization of input during web page generation ('Cross-site scripting'). The MITRE Corporation; 2006. https://cwe.mitre.org/data/definitions/79.html. [Accessed 5 May 2019].
[102] Olsen A. Determining the Tobii I-VT fixation filter's default values. 2012.

[103] Olsen A. The Tobii I-VT fixation filter algorithm description. 2012.
[104] Holmqvist K, Nystrom M, Andersson R, Dewhurst R, Jarodzka H, Weijer J van de. Eye tracking A comprehensive guide to methods and measures. New York: Oxford University Press; 2011.
[105] Kevic K, Walters BM, Shaffer TR, Sharif B, Shepherd DC, Fritz T. Eye gaze and interaction contexts for change tasks – observations and potential. J Syst Software 2017;128:252–66.
[106] Hunter JD. Matplotlib: a 2D graphics environment. Computing in Science \& Engineering 2007;9:90–5. https://doi.org/10.1109/MCSE.2007.55.
[107] Blascheck T, Schweizer M, Beck F, Ertl T. Radial transition graph. n.d. http://www.rtgct.fbeck.com/. [Accessed 1 December 2019].
[108] PlantUML. PlantUML in a nutshell. n.d. https://plantuml.com/. [Accessed 5 July 2020].
[109] Kahneman D, Beatty J. Pupil diameter and load on memory. Science 1966;154:1583–5.
[110] Goldinger SD, Papesh MH. Pupil dilation reflects the creation and retrieval of memories. Current Directions in Psychological Science 2012;21:90–5. https://doi.org/10.1177/0963721412436811.
[111] van der Wel P, van Steenbergen H. Pupil dilation as an index of effort in cognitive control tasks: a review. Psychonomic Bulletin and Review 2018;25:2005–15. https://doi.org/10.3758/s13423-018-1432-y.
[112] Krejtz K, Duchowski AT, Niedzielska A, Biele C, Krejtz I. Eye tracking cognitive load using pupil diameter and microsaccades with fixed gaze. PLoS ONE 2018;13:1–23. https://doi.org/10.1371/journal.pone.0203629.
[113] Ahlstrom U, Friedman-Berg FJ. Using eye movement activity as a correlate of cognitive workload. International Journal of Industrial Ergonomics 2006;36:623–36. https://doi.org/10.1016/j.ergon.2006.04.002.