EYE MOVEMENTS CHARACTERIZING FOR THE ASSESSMENT OF EXPERTISE IN SOURCE CODE READING

A dissertation submitted

to Kent State University in partial fulfillment of the requirements for the degree of Doctor of Philosophy

by

Salwa Aljehane

August 2022

© Copyright All rights reserved

Except for previously published materials

Dissertation written by Salwa Aljehane

B.S., Taibah University, 2007 M.S., Kent State University, 2015 Ph.D., Kent State University, 2022

Approved by

Dr. Jonathan Maletic , Chair, Doctoral Dissertation Committee

Dr. Feodor Dragan Dr. Jong-Hoon Kim Dr. Bonita Sharif

, Members, Doctoral Dissertation Committee

Dr. Jocelyn Folk

Dr. Joseph Ortiz

Dr. Javed I. Khan

Dr. Mandy Munro-Stasiuk

Accepted by

, Chair, Department of Computer Science

, Dean, College of Arts and Sciences

# TABLE OF CONTENTS











 **Research Focus 5**

 **Research Questions 6**

 **Contributions 7**

 **Organization 8**



* 1. **Source Code Comprehension 9**
     1. **Readability and Program Comprehension** 9
     2. **Expertise and Program Comprehension** 14
  2. **Expertise in Programming 17**
  3. **Eye-Tracking in Software Engineering 23**
     1. Eye-Tracking in Program Comprehension 23
     2. Eye-Tracking in Model Comprehension 26
     3. Eye-Tracking in Code Summarization 27
     4. Eye-Tracking in Debugging 28
     5. Eye-Tracking in Reading Pattern 29



* 1. **Fixations 32**
     1. Fixation Count 33
     2. Fixation Rate 33
     3. Fixation Time 34
     4. Ratio of Fixation Time 35
  2. **Saccade 36**
  3. **Pupil Size Metrics 36**
  4. **Scan Path 38**
     1. Attention Switching Frequency 38
     2. Scan Match 38
  5. **Heat Map 38**



* 1. **Dataset and Task Used 44**
  2. **Data Cleaning and Transformation 45**
     1. Source Code Data 45
     2. Eye-Tracking Data 46
     3. Navigation within Source Code 46
     4. Identify Term Location and Mapping 47
  3. **Results and Discussion 48**
     1. RQ1: Analysis at Term Level 48
        1. Expert vs. Novice 51
        2. Discussion 54
     2. RQ2: Total Gaze on Source Code Lines 55
  4. **Conclusion 56**



* 1. **Study Design and Procedure 57**
     1. [Dataset 57](#_TOC_250026)
     2. [Measures 58](#_TOC_250025)
        1. [Visual Effort Assessment 59](#_TOC_250024)
           1. [Fixation Metrics 59](#_TOC_250023)
           2. [Saccade Metrics 61](#_TOC_250022)
        2. [Cognitive Effort Assessment 61](#_TOC_250021)
           1. [Pupil Size Metrics 62](#_TOC_250020)
           2. [Measure Pupil Dilation 62](#_TOC_250019)
     3. [Evaluate Expertise 64](#_TOC_250018)
     4. [Data Cleaning and Transformation 64](#_TOC_250017)
  2. [Results and Discussion 65](#_TOC_250016)
     1. [Correlation Between Expertise and Eye Tracking Metrics 65](#_TOC_250015)
        1. [Discussion 68](#_TOC_250014)
        2. [RQ1: Summary 69](#_TOC_250013)
     2. [Metrics Between Experts and Novices 71](#_TOC_250012)
        1. [Number of Fixations Result 73](#_TOC_250011)
        2. [Total Fixation Duration Result 74](#_TOC_250010)
        3. [Saccade Length Result 75](#_TOC_250009)
        4. [Saccade Duration Result 76](#_TOC_250008)
        5. [Line Coverage Result 77](#_TOC_250007)
        6. [RQ2: Summary 78](#_TOC_250006)
     3. [Cognitive Load and Pupillary Response 79](#_TOC_250005)
        1. [Discussion 81](#_TOC_250004)
        2. [RQ3: Summary 83](#_TOC_250003)
  3. [Implications 83](#_TOC_250002)
  4. [Threats to Validity 84](#_TOC_250001)
  5. [Conclusion 85](#_TOC_250000)



* 1. **Visual Effort Metrics Result by Task Order 87**
     1. Experts’ First vs. Second Stimuli 90
        1. Fixation Related Metrics Result 90
        2. Saccade Related Metrics Result 92
        3. Line Coverage Result 92
     2. Novices’ First vs. Second Stimuli 92
        1. Fixation Related Metrics Result 93
        2. Saccade Related Metrics Result 94
        3. Line Coverage Result 95
     3. Visual Efforts Discussion 95
  2. **Cognitive Efforts Metrics Result by Task Order 96**
     1. Experts First vs. Second Stimuli 98
     2. Novices First vs. Second Stimuli 100
     3. Cognitive Efforts Discussion 102
  3. **Threats of Validity 103**
  4. **Conclusion 106**







# LIST OF FIGURES

Figure 4.1: An overview of the analysis process 47

Figure 4.2: An example of the srcML generated code when extracting operators 48

Figure 4.3: Comparing experts and novices eye movements in reading source code elements 51

Figure 4.4: Participants’ fixations distribution over source code areas 54

Figure 5.1: Participant's expertise distribution in EMIP for Java code 58

Figure 5.2: The timeline of the pupil diameter for novice (N 62) at the beginning of reading during the first trial (Rectangle. Java), where the baseline averaged to 3.61 mm from the first 15 samples 63

Figure 5.3: The timeline of the pupil diameter for expert (E1) at the beginning of reading during the first trial (Vehicle. Java), where the baseline averaged to 2.57 mm from the first 15 samples 63

Figure 5.4: Correlation matrix of the correlation coefficient result between eye-movement measurements and expertise 70

Figure 5.5: Experts vs. Novices in fixation-related metrics and saccade-related metric 73

Figure 5.6: Participant correctness level for Java code in Rectangle and Vehicle tasks 77

Figure 5.7: Experts vs. novices in pupil-related metrics 82

Figure 6.1: Experts’ eye movements comparing the first and second trial 90

Figure 6.2: Novices’ eye movements comparing the first and second trial 93

Figure 6.3: Experts’ pupil dilation comparing the first and second trial 98

Figure 6.4: Novices’ pupil dilation comparing the first and second trial 101

Figure 6.5: Comparing experts’ pupil dilation timeline when reading the first trial (red) and the second trial (blue), for (a) expert ID-1 and the baseline average is 2.57, (b) expert ID-11 and the baseline average is 3.97, (c) expert ID-24 and the baseline average is 3.87, (d) expert ID-190 and the baseline average is 3.72, (e) expert ID-51 and the baseline average is 2.52, (f) expert ID-113 and the baseline average is 3.248

................................................................................................................................... 104

Figure 6.6: Comparing novices’ pupil dilation timeline when reading the first trial (red) and the second trial (blue), for (a) novice ID-6 and the baseline average is 4.012, (b) novice ID-29 and the baseline average is 2.94, (c) novice ID-30 and the baseline average is 3.04, (d) novice ID-31 and the baseline average is 3.718, (e) novice ID-41 and the baseline average is 4.53, (f) novice ID-133 and the baseline average is 4.611

................................................................................................................................... 105

# LIST OF TABLES

Table 2.1: Overview of expertise methods in some software engineering studies 22

Table 3.1: An overview of some eye-tracking metrics used in some recent studies 40

Table 3.2: An overview of using pupil size metric in some recent studies 42

Table 4.1: Mann-Whitney results for comparing novices and experts in navigation areas.

Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d. 52

Table 5.1: The result of experts and novices on eye tracking measures 72

Table 5.2: Mann-Whitney test result of experts and novices in fixation-related metrics and saccade-related metrics. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d. 75

Table 5.3: Mann-Whitney test result of experts and novices in pupil-related metrics. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d. 82

Table 6.1: The results of the actual data comparing the visual effort of experts and novices in first and second stimuli 89

Table 6.2: Wilcoxon test results of experts in fixation related metrics and saccade related metrics comparing the first and second stimuli 91

Table 6.3: Wilcoxon test results of novices in fixation related metrics and saccade related metrics comparing the first and second stimuli. Asterisk indicates significant with 95% confidence. The effect size is given by Cohen’s d. 94

Table 6.4: The results of the actual data comparing the cognitive effort of experts and novices in first and second stimuli 97

Table 6.5: Wilcoxon test results of experts in pupil related metrics comparing the first and second stimuli. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d. 100

Table 6.6: Wilcoxon test results of novices in pupil related metrics comparing the first and second stimuli 102

**ACKNOWLEDGEMENTS**

First, I would like to express my greatest gratitude and appreciation to my advisor, *Prof. Jonathan I. Maletic*, for always believing in me and my work, while also supporting me throughout the entire process. Through his knowledge, wisdom, and valued advice the process of completing this dissertation was possible.

Second, with my deepest love and endless gratitude towards my kids and my family, who without their support this journey wouldn’t have been possible. Their continued patience and love throughout the years has been my biggest motivator while completing this dissertation. Thank you to my parents for always keeping me in their prayers because they truly have been my biggest supporters throughout it all. My sincerest love and appreciation go out to all my siblings who have been there for me since day one. A huge thank you also goes out to my whole world, my kids *Razan* and *Sultan* for their patience and support for me in this journey, I wouldn’t have done it without them.

Finally, I am also thankful to all my friends and colleagues in my department for their help and understanding in this research journey. To my dissertation committee, thank you so much for your time, feedback, and comments that helped in the completion of my dissertation.

Salwa Aljehane

July 2022, Kent, Ohio



# INTRODUCTION

Expertise is an important criterion for programmers that contributes mainly on how they perform the programming related tasks. However, years of programming experience is not the only measure that is related to expertise (Feigenspan et al., 2012). Researchers have shown that expertise reflects the performance on how programmers perform and not necessarily how long they have trained (Shanteau, 1992). This new understanding of expertise would open the door to adopting new strategies from other domains of deliberate practice (a well-designed purposeful, and guided practice) to develop expert performance in the software engineering field (Ericsson & Lehmann, 1996; Anders Ericsson, 2008). The authors claim that a fully focused practice designed by professionals that meets the following conditions would significantly affect the development of an individual expertise. Those conditions include defining the goal of the given training activity, getting appropriate and immediate feedback, and dividing the entire task into smaller sections of sub-tasks to practice instead of repeating the full task as a whole. Therefore, some approaches that follow these principles of deliberate practice could also be applied to improve programming skills with more focus and attention.

At the same time, different programming expertise levels require a different mental workload to solve a programming task (Kuric & Bieliková, 2014). Program comprehension, for example, is a cognitive process that allows developers to use their

knowledge with such a mental model to acquire information from the code and draw their conclusions (Feigenspan et al., 2012). However, a developer’s programming experience is a sensitive key to ease this involved internal cognitive process.

Over the years, researchers have made many attempts to measure programming experience and try to evaluate programmers activities with respect to their expertise (Arisholm et al., 2007; Feigenspan et al., 2012), as well as to understand developers’ behaviors during programming activities corresponding to their expertise level, such as in source code comprehension (Soloway & Ehrlich, 1984; Bednarik & Tukiainen, 2006; Feigenspan et al., 2011), maintenance (Yu et al., 2019), and debugging (Vessey, 1985; Alqadi & Maletic, 2017). There are many ways for managing expertise in conducted studies, including filling questionnaires in prior experiments (Feigenspan et al., 2011), years of programming, education level (Ricca et al., 2007; Kevic et al., 2015; Busjahn et al., 2015; Abid, Maletic, et al., 2019), and how programmers evaluate their experience (Feigenspan et al., 2012). Further research studies have been conducted by using eye- tracking technologies to study the effect of expertise levels on comprehension during code reading (Busjahn et al., 2011), reviewing (Uwano et al., 2006), and summarization (Rodeghero et al., 2014). Another eye-tracking study attempts to evaluate the differences between the expertise and professional status of software developers (Soh et al., 2012).

Due to the existence of multiple programming languages (Shrestha et al., 2020), each of which is best suited for specific applications. In addition, the need of being up to date with new technologies, and the availability and expansion in learning resources nowadays is even more important as becoming an expert is no longer limited to how long

you are at the job. Thus, new ongoing studies and methods are necessary to better measure these new levels of expertise. There is a need to clarify and redefine expertise rather than using professional status, education level, or programming years. Moreover, with fast improvements in the software engineering field, there should be a consistent improvement in developer expertise assessment. One example of this is the adoption of more realistic methods for developer expertise evaluation in a realistic environment while conducting the programming activities (i.e., eye movement tracking on code reading).

The general objective of this research is to provide a reliable approach to characterize developers’ expertise level as expert/novice via findings from their eye movement data. One way is to examine the differences between experts and novices at a finer level of granularity, namely the source code element level. For this purpose, we use multiple source code parts to evaluate the impact of expertise on navigating the code area. Through this approach, we provide an in-depth analysis of the reading process that includes statement and term levels to capture more detailed information. Our work also focuses on the impact of expertise on viewing the source code elements regardless of the time duration. This study analyzes an existing eye tracking data set to study programmers’ behaviors and strategies during source code reading in the context of a bug fix. The data was collected in 2015 (Kevic et al., 2015). Kevic et al. conduct an eye tracking study on large source code in an open-source system using the Eclipse plugin iTrace (Shaffer et al., 2015; Guarnera et al., 2018). Twenty-two programmers at two different expertise levels (experts and novices) participated in this study. The participants consisted of twelve professional programmers working in industry and ten computer science students.

In another evaluation study using a different dataset, we aim to make use of the eye movement related metrics to find a distinct pattern that can differentiate between subjects in a program comprehension task related to their expertise. To achieve this goal, we adapt multiple metrics to describe the differences between experts’ and novices’ visual behaviors when solving comprehension tasks. In the same study, we perform an empirical investigation to study the extent to which type of expertise metrics correlate the best with eye-tracking parameters. To validate the results, we choose the metrics from three main categories: fixation-related metrics, saccade-related metrics, and pupil dilation metrics. For this analysis, we utilize the EMIP distributed eye-tracking dataset (see section 5.1); this open dataset includes the gaze data of 216 developers, each of whom solved two comprehension programs (Bednarik et al., 2020).

In terms of analyzing pupillometry data before comparing subjects, one way is to predefine pupil peak values. One can then find and average the changes in the pupil sizes relative to the baseline for each developer up to each threshold. According to the Beatty reviews in 1982 (Beatty, 1982), Hakerem and Sutton provided one of the first attempts of pupillometric analyses at the visual threshold (Hakerem & Sutton, 1966). We adopted this approach which has been presented and used previously in multiple studies to analyze pupil dilation, such as in (Klingner, 2010; Fritz et al., 2014; Eckstein et al., 2017).

Pupil size is influenced by cognitive load and usually tends to dilate up to 0.5 mm above its relative baseline value (Beatty, 1982; Beatty & Lucero-Wagoner, 2000; Sirois & Brisson, 2014). Many early studies have been conducted to prove the reflection of cognitive workload effect on pupil size (Hess & Polt, 1964; Hoeks & Levelt, 1993; Marshall, 2002).

Thus, pupil dilation has been used in many studies as a measure to study mental workload (Bailey & Iqbal, 2008; Klingner et al., 2011), to explore the relationship between cognitive ability and the pupil baseline (Tsukahara et al., 2016), and to combine pupil dilation with other physiological measurements to assess cognitive load (Hogervorst et al., 2014).

This study uses the changes in the pupillary response of expert/novice developers as an indicator for the underlying cognitive efforts they performed. To support a valid dilation analysis and to perform a fair comparison between developers, this study includes developers who solve tasks in the same language (Java). On average, the results show that less experienced developers have a statistically higher average of fixations with dilated pupils than skilled developers. This result suggests that novices apply more attentional focus and mental efforts to solve the comprehension task than expert developers.

 **Research Focus**

The first goal of our study is to improve ways to assess expertise by using realistic

methods. For example, we analyze programmers’ eye movements while reading source code lines to find a distinguishable pattern that differentiates experts and novices. Thus, we study the programmers’ reading process while navigating the source code at the line and term level. The second goal is to understand which parts of the source code are important to the programmers so that they can focus on those parts more than on the other areas of the code.

Another focus of this research is to explore how years of programming experience influence the visual behaviors of developers during a comprehension process. Under this goal, we seek to measure the differences between expert and novice programmers in their

performed visual and cognitive efforts during a comprehension task. Visual effort is assessed using fixation related metrics and saccade related metrics, while cognitive workload is measured using pupil dilation metrics.

# Research Questions

To address the research goals, we seek to answer the following research questions

during two evaluation studies on two different eye movements dataset:

Questions on experts and novices eye movements over source code study:

* RQ1: Considering programmer eye movements over Java source code, how do experts and novices compare when focusing on method signatures, identifiers, types, operators, keywords, arguments, names, and types in if condition statements, else statements, and while statements?
* RQ2: Which parts of the code elements are looked at the most by the programmers (experts and novices)?

Questions on assess expertise using eye-tracking related measurements study:

* RQ1: Which is the best representative measurement for estimating expertise that shows a best connection with eye-tracking metrics?
* RQ2: Using eye movement measurements: fixation counts, total fixation duration, code lines coverage, saccade length and saccade duration, to what degree do experts differ from novices?
* RQ3: To determine the differences between the cognitive load effort of experts and novices, can a developer’s pupillary response contribute to assessing expertise?
* RQ4: Does the order of the task (order in which the stimulus programs were shown to the participants) have an impact on the eye measurements of experts and novices during the comprehension process?

 **Contributions**

The contributions in this dissertation include:

* A definition of multiple navigation areas in the source code and mapping them to the eye movement data
* A fine-grained source code element-level study of what programmers read in source code
* A comparison between experts and novices in multiple navigation areas within source code

Then, in studying developers’ eye movements-based metrics in accordance with

their expertise, the performed empirical analysis of programmers’ eye movements provides the following contributions:

* Identifying multiple eye movement-based metrics which are extracted from the EMIP dataset (Bednarik et al., 2020).
* Examine the changes in these metrics if they are influenced by the developer’s expertise level (novices/experts).
* Assessing of the differences between experts and novices in cognitive workload using a pupillary response analysis.
* Studying the relationship between multiple types of developer expertise metrics and their eye movement metrics
* Introducing an approach that utilizes developers’ eye movement features to predict expertise
* A comparison of the reading behavior on the first trial to the second one within each group of programmers (experts and novices). This includes analyzing the changes on programmers’ gaze time, the number of visits on the source code and pupil dilation with respect to the program order.

 **Organization**

The rest of this dissertation is organized as follows: Chapter 2 presents an overview

of the related work of measuring expertise in programming then focuses on eye-tracking studies on software engineering. Then, Chapter 3 gives an overview of a wide range of eye- related metrics that have been utilized and found in previous eye-tracking studies. Chapter 4 demonstrates the approach design, along with the comparison study results that compares experts and novices on reading source code constructs. Next, Chapter 5 gives an overview of the second study that utilizes the EMIP dataset, including the description of the analysis plan and the research questions. Chapter 6 follows up on the previous chapter and demonstrates the results of comparing developers’ reading behaviors in the first trial to the second one. Finally, Chapter 7 discusses the plan of work and the future direction.



# BACKGROUND AND RELATED WORK

This chapter presents background and related work related to the research topic. It begins with an overview of source code comprehension, focusing on two relevant aspects: readability and expertise. Then a considerable amount of literature is given on the role of expertise. These studies discuss the developers’ behaviors during multiple programming activities corresponding to their expertise level and look into how authors have measured the expertise in various ways. The last section gives an overview of using eye-tracking technology in software engineering research.

# Source Code Comprehension

There are multiple factors that play an important role in source code comprehension; they are either related to the code being read or to the developer who is doing the reading.

# Readability and Program Comprehension

Lawrie et al. study the influence of identifier names on program comprehension (Lawrie et al., 2007). They propose multiple questions to investigate the effects of identifier names in different aspects. The first question asks about creating high-quality names related to naming identifiers with relevant meaning. The second question asks whether using initials or abbreviations in identifier names would be enough to reflect meaningful names, or whether it would be better to use full names. The third question considers the limitation of human short memory when asking how longer names can affect retrieving

and remembering abilities. 128 subjects of different genders, education levels, and experience participated in the study. They were instructed to answer questions about identifier names in 12 different functions after describing them. The results related to the length of the names show that using long terms for naming identifiers makes them hard to remember, negatively affecting comprehension. In contrast, choosing short, meaningful identifiers or abbreviations are easier to remember by participants and are therefore better for assessing the comprehension process. Additionally, their results provide further support for the hypothesis that tested the impact of experience and education on comprehension. In terms of understanding lower quality identifiers, the decrease in experience and education factors negatively impact the comprehension of these identifiers compared to more experienced and educated subjects; the result is significant. However, experience and education do not contribute significantly to writing correct function descriptions.

The same findings provided later in 2017 relate specifically to the impact of using short identifier names on comprehension. Hofmeister et al. investigate the effects of naming identifiers (either using letters, words, or abbreviations) in two main aspects: length and meaning (Hofmeister et al., 2017). Seventy-two professional subjects experienced in C# were recruited to participate in the study. Each participant was asked to review six code snippets to find one defect in each code and then provide a solution; three have semantic defects, and the three others have syntax errors (each snippet has a different naming style). The process was repeated several times to find the defects and failing to locate them resulted in excluding that subject. Hofmeister et al. show developers’ comprehension efforts to perform the task utilizing the time needed to find the semantic defects. Therefore,

their findings confirm the association between the comprehension process and the length/semantics of identifiers. The results find a significant difference between the time needed to comprehend words against non-words (letters and abbreviations) with a medium effect size. However, the observed difference between the time needed to comprehend letters against abbreviations is insignificant. Additionally, the result did not show any significant impact of the identifier names on the comprehension time while locating the syntax errors.

On the other side, Fakhoury et al. investigate the effects of using poor source code identifiers and comments on code readability, consequently impacting the code comprehension (Fakhoury et al., 2018). Specifically, using a brain monitoring technique (fNIRS), they empirically study the impact of identifiers quality on the cognitive load needed to comprehend the code. Their experiment was conducted using a combination of fNIRS and eye-tracking devices in order to capture the cognitive effort devoted to understanding the source code. 70 graduate and undergraduate students were participants in this study, all of whom were asked to perform four comprehension tasks, each with 30 to 40 LOC from open-source projects. The changes in the oxygenation concentration (Oxy) are used as an indicator of the cognitive load experienced during the comprehension study. Then, they analyze the mapping between the fixations on the source code (identifiers) and the cognitive load. One of the main results of the work by Fakhoury et al. shows the association between source code readability and the developer exhibiting a high cognitive effort. Moreover, they provide evidence that the use of a poor structure of the source code

(poor identifiers and comments) would reflect mainly on increasing the cognitive effort while reading the programs.

In another recent study, Couceiro et al. propose multiple ways to investigate the relationship between the cognitive effort of programmers and the code complexity, such as using biological signals obtained from EEG, ECG, and eye-tracking devices (Couceiro, Duarte, et al., 2019a). So far, this method has been applied only by using the changes in the programmer’s heart rate as an indicator of the devoted mental effort toward comprehending a code with different complexity levels. They conducted their experiment by monitoring the HRV (heart rate variability) of 26 participants who were asked to read three Java programs with various levels of complexity categorized based on using the NASA-TLX assessment test. The result of analyzing the ECG signals of subjects shows that using HRV is a suitable method to identify the increase in the subject’s mental effort while understanding the code. Also, Couceiro et al. find that applying code complexity metrics (such as nested block and LOC) is inaccurate in measuring the cognitive effort needed and how programmers perceive the tasks (easy/difficult).

Going into more detail about source code readability, Scalabrino et al. propose some readability models that include multiple textual features which need to be considered when measuring code readability (Scalabrino et al., 2016). However, adding to the previous findings that use only structural features in measuring source code readability (such as lines of code), Scalabrino et al. provide more textual features to utilize in source code readability characterization (such as identifiers and comments quality). These features were selected based on the relationship between using high-quality source code comments and identifier

names and improving the comprehension process. The readability models (logistic regression classification) were built and trained using three different data sets, which have, in total, more than 5000 subjects who manually assessed the readability of 600 code snippets. The results show that the combination of the textual and structural source code prosperities would increase the model accuracy of measuring the readability of the code.

Similarly, the findings in the Wulff-Jensen et al. study further support the idea that the structural and textual code properties’ effects on source code readability can reflect comprehension (Wulff-Jensen et al., 2019). As the purpose of the study is to examine readability, the fixations of 21 participants were collected during an eye-tracking study while reading four code snippets. After reading the code, subjects were asked to answer two different questionnaires concerning their ability to comprehend it and their expertise, then write a summary explaining their understanding of the code. The increase in the fixation duration metric was used as an indicator that missing structural/textual features have a negative impact on code readability. Like the findings in Scalabrino et al. but using different technology, Wulff-Jensen et al. conclude that including textual code features can result in a better comprehension process. This result was noted after recording a significant increase in the average gaze time over the source code with no textual features compared to the average time spent reading other code snippets. There is also a decrease in the accuracy of the summaries (more mistakes) with missing textual features. This result shows the importance of containing textual code properties that support the understanding of the code. However, the researchers noticed that the lack of structural features in the code result in a noticeable increase in the fixation count and the saccade length metric.

# Expertise and Program Comprehension

Scalabrino et al. address different questions concerning quantifying understandability empirically (Scalabrino et al., 2017). They identify three main characteristics to investigate: metrics related to developers, metrics related to documentation, and metrics related to the code. Their experiment includes 46 developers with multiple expertise levels who were asked to understand 6 code snippets in Java and then answer three comprehension questions about the code. However, understandability was measured by design variables related to the time needed and the programmers’ comprehension of the task (i.e., participants choose the answer between understand or cannot understand the code). By testing the correlation between understandability variables and other metrics, they noted that none of the defined metrics could form a relatively good correlation with the understandability. Focusing mainly on developer experience metrics, the result shows that there is a weak correlation between the amount of time needed to comprehend the code and the years of experience in the Java programming language. Also, developers with higher experience tend to perform better on the actual understanding metric, which tests the correctness accuracy for the programming comprehension tasks, than the less experienced subjects.

Recently, Peitek et al. introduce a new method to measure understandability using a brain imaging technique (fMRI) (Peitek et al., 2020). Specifically, in their study, they use the BOLD signal (blood oxygenation level dependent) in order to demonstrate the activation in the brain areas during the comprehension process. Seventeen students were recruited for the study. Then a replication experiment was conducted on 11 more students

for confirmation purposes. The subjects’ expertise was determined using a self-evaluation method on general programming and on the specific programming language used in the experiment (Java). On the question of the role of experience on the recorded brain signals, the results of the study show that general programming experience does not influence lowering the brain activation (the cognitive effort). However, the results of the activation patterns reveal that the level of experience in Java correlates negatively with cognitive efforts. These findings further support the association between the experience factor and the performed mental efforts during the code comprehension process.

Likewise, Floyd et al. have highlighted the relevance of expertise to conducting programming activities using the same experimental method explained in the Peitek et al. study (fMRI) (Floyd et al., 2017). Their study argues that analyzing developers’ brain activities is a promising technique that can reveal insights into the impact of individual experiences and observed differences between reading code vs. natural languages. They conducted their experiment by monitoring the brain activation of 29 participants with various levels of expertise while reading code or prose stimuli. They use participants’ GPAs as an indicator of their programming skill levels. The process starts with alternating between three types of tasks presented to the subjects: code review, code comprehension, and prose review. Then, three types of binary classification models were implanted to distinguish between code and prose using the collected fMRI dataset. Finally, their accuracy results are compared. The results show that the Gaussian Process Classifier (GPC) has a high predictive power across all models with 79% accuracy. Related to expertise, the authors find that the analyzed brain activation patterns to distinguish between (code vs.

prose) are less distinct in skilled subjects. Thus, unlike novice developers, experts put less cognitive effort toward comprehending source code, similar to comprehending prose.

In the same year, a study by Siegmund et al. is conducted also using the fMRI technique to view comprehension mental models in terms of which region of the brain activated during the understanding process (Siegmund et al., 2017). The results show that the fMRI data provide useful identification and are able to differentiate between the brain areas involved during two mental process: bottom-up comprehension and comprehension with using semantic cues.

Another important aspect that has to be mentioned is the impact that expertise has on the ability of code comprehension. Previous research by Ricca et al. has attempted to explain the role of using the UML diagram vs. Conallen’s stereotyped diagram in program comprehension (Ricca et al., 2007). The effects have been studied in three subject groups with multiple experience levels; the authors use education level (undergraduate/graduate) to refer to expertise (low/high). In three individual experiments, Ricca et al. conducted their study with 66 participants who were asked to visualize diagrams and then answer 12 comprehension questions to explain their understanding. A statistical comparison of the two expertise groups reveals that in the undergraduate group (novices), stereotyped class diagrams support the understanding process more than in the expert group. However, the graduate students (experts) rely on the code and spend more time visiting the code than visualizing the diagrams.

To determine the effects of expertise, Lee et al. compare the EEG signals obtained from expert and novice individuals while comprehending 36 programming tasks written in

Java (Lee et al., 2016). The 18 participants in the study are characterized into the two groups based on their coding skills in Java (the experiment programming language) and an expertise self-evaluation method. Specifically, prior to their experiment, subjects were given a comprehension task where the response time and correctness were evaluated to divide the participants based on their skills. The solving time of task shows better accuracy in differentiating the two groups than correctness. Through an empirical investigation of the neural representations of experts and novices, Lee et al. are capable of distinguishing program comprehension abilities from participants with different programming experiences. They find that experts recorded higher brainwaves activated in multiple brain regions by utilizing more cognitive skills than novices. This finding suggests that expert programmers handle the comprehension process successfully with higher short-memory ability and better skill in retrieving code functionality afterward than novices.

Together, these studies outline the important factors in code comprehension that concern either source code-related issues or the key role of experience in increasing and explaining a developer’s understanding.

# Expertise in Programming

Over the years, researchers have made many attempts to measure programming experience and try to evaluate programmers’ activities with respect to their expertise (Arisholm et al., 2007; Feigenspan et al., 2012; Siegmund et al., 2014), as well as to understand developers’ behaviors during programming activities corresponding to their expertise level, such as in source code comprehension (Soloway & Ehrlich, 1984; Feigenspan et al., 2011; Bednarik & Tukiainen, 2006), maintenance (Yu et al., 2019), and

debugging (Vessey, 1985; Alqadi & Maletic, 2017). The following briefly demonstrates some findings of these studies:

In an analysis of programming experience, Feigenspan et al. examine multiple ways to define and control expertise in conducting programming experiments (Feigenspan et al., 2012). Using the previous research work, they provide a range of important metrics to access expertise, including four categories: years of programming, level of education, self- estimation, and size of programs that the subject is able to write. In order to testify the accuracy of each measure, they designed a questionnaire with multiple questions based on each category. 123 participants were chosen to complete the programming experience survey; the subjects were selected from three universities with different levels of education. Each was asked to solve ten program comprehension tasks. Correctness and response time solving the tasks were the selected measures to evaluate the influence of the chosen expertise metrics on comprehension. Their analysis of programming experience found evidence that self-evaluation among the undergraduate group is a key indicator for explaining programming experience.

In further analysis of expertise, Arisholm et al. conduct an empirical study to examine the emerging role of pair programming in the context of improving programming quality and correctness (Arisholm et al., 2007). These effects have been studied for a different level of expertise and task complexity while participants perform six change tasks. Their experiment was conducted in two phases with 99 individual developers and then 98 pairs. The expertise among participants was decided based on the results of a pretest programming task and the developers' professional qualification degree as (junior,

intermediate and senior). The set of the changing tasks on two Java systems consists of several classes. The study's results show an increase of correctness in the junior and intermediate programmer groups, while the correctness percentage decreases in the senior group compared to that of individuals. However, duration results report a 39 % decrease in time for the intermediate group (in pair programming). In comparison, there is a slight decrease of 8 % in the duration of senior pair programmer groups compared to other participants.

Soloway and Ehrlich introduce programming plans when separate program statements bind together to build a related goal (Soloway & Ehrlich, 1984). The study uses this notation to empirically investigate the role of programming experience in the cognitive comprehension process and the interpretation of the Pascal program. A total of 139 students participated in the study (94 novices and 45 experts). Expertise was evaluated in the experiment based on the students’ level in the programming class. Then, the data were collected through two types of studies: using a plan-like program and an unplan-like program. Two independent variables are measured to test the expertise effects on the cognitive comprehension model: the total time to finish the task and the correctness of the given comprehension questions. Results show that experts are better at telling when program rules are violated, as reflected in their performance. In addition, experts use knowledge of the programming style conventions during comprehension on an implicit level.

In contrast, Bednarik and Tukiainen use eye-tracking technology to study the program comprehension process in the context of an expertise level (Bednarik &

Tukiainen, 2006). Their approach uses a program visualization technique to study the programmer's behavior and track any developments in the comprehension process. A total of 18 students ranging from low to intermediate programming levels participated in the study. Expertise in this experiment was characterized based on the education level: high school students who attend college programming courses are indicated as novices, undergraduate and graduate computer science students are characterized as intermediate programmers. Each subject was asked to visualize three Java programs along with the program code lines represented as text. Analyzing the comprehension time captured by the eye tracker in the two code representations shows that less experienced developers pay more attention to the visualization to aid their comprehension at the beginning of the experiments. However, the duration time decreased toward the later stages. On the other hand, more experienced developers start looking at the code and spend more time in the code than the visualization, which explains that they use it to gain additional information about the code.

In the debugging domain, Alqadi research analyzed novices' behavior in terms of the time response and the correctness level during the debugging process (Alqadi & Maletic, 2017). The study involved 142 computer science students taking introductory programming classes. To identify the role of programming experience in the process of debugging, they conducted their experiments in two phases over two years. The first experiment aims to study the confident level of identifying the code's logical errors and matching them to the corresponding type. In the second experiment, the study investigates the understandability of code errors and the ability to provide a correct solution. The study

findings confirm the association between the level of expertise and the ability to find the errors in a proper time. Particularly in the first study, experiment results conclude that programmers with higher programming experience spent a short time locating the code errors. However, in the second experiment, more experienced developers are better at providing correct answers to fix the code**.**

# Measuring Expertise

To date, there has been little agreement on the most accurate metric to use for expertise. Several methods have been used in the previous research to control the expertise factor in the study. However, it is an important issue that should be investigated due to the direct impact in establishing the reliability of the experiments to draw a dependable conclusion. There are many ways for managing expertise in conducted studies, including filling out questionnaires in prior experiments, years of programming, education levels, and how programmers evaluate their experience.

The following Table 2.1 provides a brief example of these multiple expertise evaluation methods used in some previous software engineering studies and demonstrates how they are investigated (Feigenspan et al., 2012; Siegmund et al., 2014).

# Table 2.1: Overview of expertise methods in some software engineering studies

|  |  |  |
| --- | --- | --- |
| **Measures** | **Method** | **Studies** |
| **Years of programming** | The total years of practicing programming in general, in a certain language or a specific domain. | (Sillito et al., 2008); (Fritz et al., 2014); (Scalabrino et al., 2017) |
| **Education level** | Using education level as an indicator for programming experience such as: Graduate/ undergraduate, faculty/ student, or student/ professional developer | (Kevic et al., 2015); (Busjahn et al., 2015); (S. C. Müller & Fritz, 2016); (Abid, Maletic, et al., 2019); (Ricca et al., 2007) |
| **Self-evaluation of experience** | How developers evaluate their programming experience: none, low, medium, high, or on a scale level from 1 to 10 | (Feigenspan et al., 2012); (Peitek et al., 2020); (Lee et al., 2016); (Peitek et al., 2020) |
| **Code size** | The size of programs developer has written, categorized into small, medium, and large | (M. M. Müller, 2004);  (Feigenspan et al., 2012); (Lee et al., 2016) |
| **Pretest** | Giving subjects a pretest to assess their programming expertise, then assign subjects into three experience levels: little, medium, and excellent | (Biffl & Grossmann, 2001); (Arisholm et al., 2007); |

* 1. **Eye-Tracking in Software Engineering**

This section describes and discusses the previous work on using eye tracking technology in software engineering studies. Eye trackers collect developer’s eye gaze data on a visual display while performing a given programming task. Relatively, during eye tracking experiments, looking at the screen, would trigger such a mental process for understating the given tasks. In this way, we can say that visual efforts play an important role as a proxy to the cognitive process. Thus, the gaze data holds a lot of valuable insights that could help to understand the developer reading behaviors and comprehension ways. This would explain the benefit of using eye tracking technology in software engineering studies in a practical way.

Eye tracking in software engineering has been explored in multiple studies using a variety of techniques and methods. The common categories that have multiple efforts to address the use of eye tracking technology are code comprehension (Sharif & Maletic, 2010a), model comprehension (Sharif, 2011; Sharif & Maletic, 2010b), debugging (Bednarik, 2012), and code summarization (Rodeghero et al., 2014; Rodeghero & McMillan, 2015). The following summarizes and demonstrates the findings of several eye tracking studies.

# Eye-Tracking in Program Comprehension

The study of eye movements is gaining popularity in different software engineering domains. Crosby and Stelovsky conducted one of the first studies to use eye tracking (Crosby & Stelovsky, 1990). The study includes 19 subjects: 10 with low programming experience from CS 2 class, and 9 graduate students with high experience. All been asked

to read a binary search algorithm that written in Pascal language. The goal of their study is to investigate the impact of experience and the viewing strategies when reading an algorithm. Crosby and Stelovsky show that reading source code is different than reading natural text in which programmers need more fixations while reading the algorithm than they do when reading the text. They also found that experience is an essential impact on the subjects when focusing on the main area of the code.

Crosby et al. also conducted an eye tracking study later to understand the influence of experience level on using beacons while reading code (Crosby et al., 2002). Beacons is the surface features simplify the program comprehension which first studied and discussed by Brookes (Brooks, 1983) of the use of the most expressiveness segment of the code. Three experiments were conducted with programmers in two different expertise levels (expert and novices). First experiment is to study the level of understandability toward program functions. They conducted this experiment with 45 participants who were asked to rate lines of code on a scale of 1 to 6 (binary search code mixed with another distracter code) as to how likely a line of code represents a binary search algorithm. The second experiment measures the participants time response and the answer correctness. There are 30 programmers in this experiment who were given lines of code from binary search code, shell sort code and depth first search. Each had been asked to choose which lines of code represent which program. Third experiment was dedicated to determining the time and the exact statements that participants look at precisely using eye tracking (19 programmers). The result of the study shows that the use of beacons is different between experts and novices. More experienced programmers tend to focus on the important area of the code

and used beacons to understand the program than novices. Novices made little distinguish between the areas of the code while experts used the complex statements as beacons. Also, the results suggest that the more experience programmers are the more they able to distinguish program type and to identify the code lines correctly.

Turner et al. compare the students’ gaze data in two programming languages, Python and C++ (Turner et al., 2014). The analysis addresses the role of the programming languages in code comprehension for novice students and non-novice students. They conclude that the accuracy of solving the tasks is higher for C++ than for Python. Also, they confirmed that Python code is associated with grater fixation duration than C++, while novices need more fixations to solve the C++ task.

To facilitate programmer work and to help them avoid introducing bugs into the code, Fritz et al. introduce a new approach to determine task difficulty by using an Electroencephalography (EEG) sensor and an eye- tracker (Fritz et al., 2014). The main goal of the study is to predict how the programmer perceives the comprehension task as easy or difficult to provide successful instructions at the right time. They conduct the study with 15 professional programmers; each of them is asked to perform 10 code comprehension tasks that are written in C#. They developed and tested their classification model on WEKA (Hall et al., 2009) using eye tracking attributes with EEG attributes. The result shows that the trained Naive Bayes classifier can predict the level of difficulty (easy or difficult) for a new programmer with 64.99% precision and 68.58% recall. Also, they conclude that blink rate and pupil dilation are the critical indicator for developers perceiving a difficult task.

# Eye-Tracking in Model Comprehension

Guéhéneuc investigate the comprehension of UML class diagram through an eye tracking study (Guéhéneuc, 2006). In this study, they also introduce a new visualization method to present the subject fixations and saccades on the diagram for the analyses purpose. They perform their eye tracking experiment with 12 graduate students who navigate two different class diagrams from two programs. After browsing the diagrams, subjects have been asked to answer one question to test their comprehension. Then, they aggregate the fixations and saccades by the visualization technique to visualize subjects most focused are (AOI) on the class diagrams. The findings show that software engineers tend to scan the class diagram first before focusing on the important parts toward comprehension the program. However, researchers noticed that subjects did not follow the relationships among classes such as inheritance.

Another study tested the effectiveness of using UML diagram characteristics on comprehension (Yusuf et al., 2007). Multiple diagram characteristics have been identified in the study, such as layout, color, and stereotype, that support the comprehension process. The eye tracking data was collected for 12 students and faculty members who read UML class diagrams of the HippoDraw open-source software. Nine subjects are experts in using UML diagrams, while three subjects have no experience in using UML (not Computer Science students). Each subject was given 27 comprehension questions to answer while visualizing the UML diagrams. The results show that experts use the diagram characteristics (coloring, stereotype provided information, and layout) more than novices when navigating the diagram to enhance the cognitive process. Also, novices tend to read

the diagram in order top to bottom and left to right compared to experts who brows the diagrams from center to the edges.

Sharif and Maletic study the role of UML class diagram layouts on comprehending the design pattern roles that tested by accuracy, time to identify the role, and the visual effort (Sharif & Maletic, 2010b). They used four design patterns from three different open- source systems, and each represented in two layout schemes orthogonal layout and multi- cluster layout. In performing their approach, the eye movements were collected during the study for fifteen participants: thirteen students (six graduate and seven undergraduate) and two faculty members. They conclude that using a multi-cluster layout has significantly improved the system comprehensibility including a higher accuracy of identifying the role in less detecting time and less visual effort.

# Eye-Tracking in Code Summarization

In 2019, a research study conducted by Abid et al. shows the role of expertise in code summarization tasks (Abid, Maletic, et al., 2019). They conduct an eye tracking study to predict the cognitive model that programmers follow when reading code for summarization: top- down vs. bottom-up models. Programmers (novices and experts) mostly read the Java methods by using the bottom-up model rather than the top-down model. Also, their results show that novices have a greater fixation duration performing the bottom-up model reading than experts. The same dataset is further analyzed by Peterson et al. to study the dwell time distribution on the line level of the source code (Peterson et al., 2019). However, they examined the effect of line length and types on the total time duration that programmers need to comprehend a line. They found that the line length is

not a relevant factor in the dwell time. Regarding all defined line types, experts and novices perform the same on the total fixation duration they spend looking at the lines.

A more detailed analysis has been done also by Abid et al., (Abid, Sharif, et al., 2019) that replicates and overcomes the limitation in the prior eye-tracking study conducted by Rodeghero et al. (Rodeghero et al., 2014). In this study, Abid et al. (Abid, Sharif, et al., 2019) compare the developers gaze behaviors collected in a more realistic environment using the iTrace eye tracking infrastructure (Shaffer et al., 2015; Guarnera et al., 2018). They show that on summarization task, developers tend to read more on the method body than just the signature. Also, while increasing the method size, experts show a significantly higher gaze when revisiting the method body than when revisiting its signature. With their findings for both developer groups, they conclude that the call terms are the most focused location for programmers in reading the code during summarization activity, followed by control flow terms then method signatures.

# Eye-Tracking in Debugging

Eye tracking studies provide insights on how programmers use scanning source code (Uwano et al., 2006) to find defects (Sharif et al., 2012). Uwano et al. shows that the longer programmers read the code the better they can find the defects. The same result is confirmed later in further research by Sharif et al (Sharif et al., 2012). In their approach, they perform an eye tracking study to explore the impact of scanning time on detected defects in source code. They show that there is a relation between scanning time and defect detection time. It is determined in the study that the longer a programmer can take on the

initial code scan, the faster they find the defects. Spending less time on the scan process also decreased the review performance.

Bednarik studies the effects of expertise in program debugging strategies (Bednarik, 2012). This study investigates the role of visual attention in debugging strategies and how that differs between novices and experts. Three visual attention switching methods are available for the participants: switching between the code and the visualization, between the visualization and the output, and finally between the code and the output. They conclude that the level of expertise is reflected in the number of bugs found. Experts find more bugs, and they spend more time in code than novices but less time in the program visualization. Novices spend more of their effort in the graphical representation of the code than the experts.

Kevic et al. conduct an eye tracking study to investigate how programmers work and navigate through three bug fix tasks (Kevic et al., 2017). Twenty-two expert and novice programmers were recruited. Twelve professional developers from industry and ten computer science students each were asked to work in three different change tasks. They found that programmers only focus on a few lines of the methods, and they spend most of the gaze time looking at variable declaration and method invocations. Also, for switching between methods, programmers rarely follow the call; instead, they mostly switch to the close element within the class.

# Eye-Tracking in Reading Pattern

Busjahn et al. conducted a study of programmers during a pro- gram comprehension task to examine the linearity of reading source code compared to natural next (Busjahn et

al., 2015). In such an approach, they study experts and novices reading source code and compared it to reading natural language text. They proposed an eye tracking study of 14 novices (students who attended Java course for beginners) and 9 experts (professional software engineers worked at companies). In the study (Busjahn et al., 2015), they use eye movement-based metrics to compare between novices and experts programmers on reading source code and how their reading style is close to linearity. Their method was inspired by the linearity which usual follow while reading natural language text. Thus, they test linearity as how closely subject follow a text’s natural reading order: top-to-bottom and left-to-right. The results show a significant difference between programmers using linearity order while reading code. The non-linearity skills of reading source code increased with expertise. They also conclude that experts cover less source code than novices and they are better to decide where look next at the code.

# At Statement and Token Level

Other researchers have done comparison studies in more fine-grained analysis on how programmers read debugging tasks. As such, Aljehane at el. (Aljehane et al., 2021) show the differences between experts and novices in reading the source code elements. Considering all the identified source code parts in the study, experts show the ability to comprehend and finish the task using fewer source code elements than the novices. The study’s results also show that the differences are significant when reading keywords, method signatures, identifiers, and variable declarations. Moreover, novices have significantly higher gaze visits when looking at more details such as names and operators in if-then-else statements.

In terms of providing analysis at the token levels, a recent work by Madi et al. (Al Madi et al., 2021) focuses on studying the effects of token frequency and length in reading source code. They demonstrate the effects by using eight eye movement metrics for each studied source code token. They found statistical evidence for the differences between low and high frequency tokens in the gaze time and total duration measurements, especially in the novices group. In addition, their results prove that the length of the source code element has a significant effect on the duration nodded to read over tokens for novice programmers.



# EYE-TRACKING METRICS

Due to the availability of eye trackers, researchers in software engineering perform eye-tracking studies for many purposes. An eye tracker records developers’ eye movement data on the visual display while performing a given task. However, tracking eye gaze behavior generates a large dataset with multiple attributes. After all, researchers can utilize this collected data for interpreting the cognitive process involved in the eye movements data. There are two significant and most widely used classifications for eye movement data: fixations and saccades; both are used to indicate the location of the participant’s attention on the visual stimuli.

In the following, this chapter summarizes three main categories of eye-tracking measures that have been explored in multiple software engineering studies (Sharafi, Soh, et al., 2015; Sharafi, Shaffer, et al., 2015; Sharafi et al., 2020). Lastly, this chapter presents an overview of the findings in some recent studies using eye-tracking metrics for the purpose of measuring visual and cognitive efforts.

# Fixations

Fixation is defined as the gaze temporarily focusing and stabilizing on one location over the source code elements for some time. Here a set of fixation-based metrics are discussed.

# Fixation Count

The fixation count metric is the total number of fixations devoted to processing one area of interest or the whole visual stimulus. This metric has been used and exported in different ways to assess the subject’s visual efforts. For example, one of the earliest studies done by Goldberg and Kotval used eye-tracking technology in the software engineering field; in this study, fixation counts served as an indicator for search efficiency: the higher the fixations amount, the more effort performed to find the relevant information (Goldberg & Kotval, 1999). In addition, fixation counts provide insights about the developer’s attention that is devoted to a specific AOI (Crosby et al., 2002; Poole et al., 2005) that would help to identify the beacons for comprehension among the program (Crosby & Stelovsky, 1990). Similarly, in Yusuf et al., they use fixation counts as a variable to indicate the cognitive efforts devoted to UML diagram comprehensibility (Yusuf et al., 2007); the higher the fixations, the more effort required to explore the diagrams. Thus, fixation counts metric defined by the following equation:

𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐶𝑜𝑢𝑛𝑡 = 𝑇𝑜𝑡𝑎𝑙 𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐶𝑜𝑢𝑛𝑡𝑠 i𝑛 𝐴𝑂𝐼 𝑜𝑟 𝑆𝑡i𝑚𝑢𝑙𝑢𝑠

# Fixation Rate

(3.1)

Fixation rate is the ratio of fixation counts in one AOI to the total number of fixations in the stimulus or to the other AOIs. It was founded by Goldberg et al. in 1999 (Goldberg & Kotval, 1999). It has been utilized in multiple studies either to measure the participant’s visual attention to a specific area (Binkley et al., 2013; De Smet et al., 2014) or to test the difficulty of an AOI that received a higher ratio of fixations compared to the

other parts of the code (Binkley et al., 2013). Fixation rate can also provide information on studying the most used source code elements while reading the code for both summarization (Abid, Sharif, et al., 2019) and debugging (Aljehane et al., 2021). Therefore, the fixation rate can be defined formally by equation 3.2:

𝐹i𝑥𝑎𝑡i𝑜𝑛 𝑅𝑎𝑡𝑒 = 𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐶𝑜𝑢𝑛𝑡𝑠 i𝑛 𝑂𝑛𝑒 𝐴𝑂𝐼

𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐶𝑜𝑢𝑛𝑡𝑠 i𝑛 𝑆𝑡i𝑚𝑢𝑙𝑢𝑠

(3.2)

A set of fixation duration-based metrics is discussed below:

# Fixation Time

Fixation time is the total gaze duration calculated either in one AOI or the whole stimulus. It reflects the individual cognitive efforts distributed on different AOIs of the visual task. However, fixation duration has been used to gain insight into developer performance and their overall visual attention that required to work on the task (Bednarik, 2012; Soh et al., 2012). A longer gaze time means that participants spend more time understanding and analyzing the task (Cepeda Porras & Guéhéneuc, 2010) due to code complexity (Busjahn et al., 2014) or defect difficulty (Cagiltay et al., 2013). Using long gaze time also refers to the more cognitive effort needed to explore the whole given task (Sharafi et al., 2013). The fixation time can be defined formally by equation 3.3:

𝐹i𝑥𝑎𝑡i𝑜𝑛 𝑇i𝑚𝑒 = 𝑇𝑜𝑡𝑎𝑙 𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐷𝑢𝑟𝑎𝑡i𝑜𝑛𝑠 i𝑛 𝐴𝑂𝐼 𝑜𝑟 𝑆𝑡i𝑚𝑢𝑙𝑢𝑠

(3.3)

# Ratio of Fixation Time

This metric is similar to the fixation rate metric, but it uses gaze time instead. This measure was proposed in 1999 by Goldberg et al. (Goldberg & Kotval, 1999) to study the influence of interface quality (good/poor) on eye movements in a programming task. The longer ratio of fixation time implies more time spent on processing the visual components (AOI) compared to the whole duration spent on reading the stimulus. It is applied later in multiple eye-tracking research work to collect insights into the software engineers’ visual efforts on programming activity. For example, researchers utilize the fixation time ratio to study the impact of the source code’s token length on eye movements (Al Madi et al., 2021) and the influence of the token type on the dwell time (Busjahn et al., 2014; Peterson et al., 2019). Another purpose of using the ratio of the fixation time metric is to study the importance of an AOI to the individuals (Bednarik & Tukiainen, 2006). It is also used to quantify the cognitive efforts devoted to understanding the source code phrases (Binkley et al., 2013) that might increase due to the higher level of the task difficulty (Cagiltay et al., 2013). In addition, Petrusel et al. investigated the effect of the time spent on the relevant region in the process model in providing the correct answers to the given comprehension questions (Petrusel & Mendling, 2013). They find that the more time spent on reading the relevant region, the higher the probability of comprehending the model. Equation 3.4 shows the ratio of the gaze time in one AOI to the time spent in the whole visual stimulus.

𝑅𝑎𝑡i𝑜 𝑜8 𝐹i𝑥𝑎𝑡i𝑜𝑛 𝑇i𝑚𝑒 = 𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐷𝑢𝑟𝑎𝑡i𝑜𝑛𝑠 i𝑛 𝑂𝑛𝑒 𝐴𝑂𝐼

𝐹i𝑥𝑎𝑡i𝑜𝑛 𝐷𝑢𝑟𝑎𝑡i𝑜𝑛𝑠 i𝑛 𝑆𝑡i𝑚𝑢𝑙𝑢𝑠

(3.4)

# Saccade

A saccade is the quick eye movement that happens between each pair of successive fixations (focus on the screen). Busjahn et al. propose two types of saccades: regression saccades represent the backtracking eye movements and forward saccades (Busjahn et al., 2015). Both types are used in this study to assess the linearity of participants’ eye movements in code reading. By analyzing eye movements using regression saccade metrics, we can characterize novice developers by using more regression saccades compared to experts. Adding to the role of regression rate in identifying expertise, a higher rate of backward saccades could also be a critical indicator of difficulty and confusion in understating a task (Poole & Ball, 2005). However, saccade-based metrics are expressed in studies by considering three factors: saccade number, saccade length, and saccade duration. Saccades are relatively quick and short, lasting for 20 – 40 milliseconds.

# Pupil Size Metrics

When the pupil dilates, it allows for more light in the eye. However, dilation occurs in different conditions, such as in low light situations or as a result of changes in the cognitive efforts of the individual (Goldinger & Papesh, 2012; Eckstein et al., 2017). Thus, instead of using the exact pupil diameter in the study’s analysis, researchers tend to use pupil dilations to measure cognitive efforts. Pupil dilation metric can provide a sensitive index to reflect the cognitive challenges when working on a difficult task (Klingner et al., 2011; Fritz et al., 2014; Kahneman & Beatty, 1966).

# Pupil Dilation and Cognitive Load

Researchers use eye movement metrics in studies to gain deep insights and understand the ongoing cognitive process while solving programming problems. For these studies, pupil size measurement is the most commonly used metric to assess the mental workload (Beatty, 1982; Hoeks & Levelt, 1993; Iqbal et al., 2004). One of the earliest such research studies was released in 1982, when Beatty conducted an empirical study reviewing multiple datasets from different domains (Beatty, 1982). The study concludes that pupil dilation is a valid indicator of the cognitive workload applied when performing a mental process. In comparison, Klingner proposed a new method to study the quick cognitive load changes by aligning the gaze events with pupil dilation (Klingner, 2010). This method allows performing an analysis of participants’ pupillary responses in a short time when understanding the visual tasks. This alignment has also successfully captured the variation that happens in pupil diameter sizes while subjects are changing between viewing multiple types of tasks.

Further research also found the same connection and concluded that there are many advantages of using pupil response as a cognitive workload measurement during interactive tasks (Iqbal et al., 2004), driving (Palinko et al., 2010), and in real-time during web browsing (Jimenez-Molina et al., 2018). However, some studies also use pupil dilation along with a measurement of blinking rates to investigate the performed mental effort (Holland & Tarlow, 1972; Holland & Tarlow, 1975; Telford & Thompson, 1933).

# Scan Path

The scan path represents the participant’s eye movement pattern. This includes the order of eye movements over the area of interest (AOI). A set of scan path-based metrics are discussed below.

# Attention Switching Frequency

This metric is computed by calculating the number of switches between a set of AOIs. It happens when the participant’s attention changes between viewing multiple AOIs.

# Scan Match

Researchers use this metric to calculate the similarity score between two scan paths. A comparison of scan paths uses the Needleman-Wunsch algorithm used in bioinformatics to compare DNA sequences. For example, (Busjahn et al., 2015) use this metric to indicate the matching score between the participants’ gaze and the reading story order, line by line from top to bottom. A high matching score between the two sequences implies that the participant reads source code much closer to how they read normal text.

# Heat Map

A heat map is one of the visualization techniques that have been utilized in different eye-tracking studies to aggregate and present the eye movements data (Sharafi, Soh, et al., 2015). Therefore, it provides a better understanding of developers’ strategies and behaviors while performing the programming task and aid the analysis of eye-tracking data. Usually, a heat map is utilized to perform a graphical representation of the fixations (the number of fixations and their duration) on top of the AOI to highlight the exact location the participant

viewed. For example, Sharif et al. (Sharif & Maletic, 2010b) use a heat map visualization to analyze the influence of using different layouts (multi-cluster and orthogonal layout) on the design pattern comprehension. In addition, Ali et al. use heat maps in their study, visualizing subjects’ fixations time over the source code to identify the most used source code elements in a programming activity (Ali et al., 2015).

Next, Table 3.1 and Table 3.2 present an overview of some recent studies using eye-tracking metrics for multiple purposes. Table 3.1 shows the findings and goals of using eye movement measurements in some research work, which fall under three categories: using fixations, saccade, or heat-map metrics. Table 3.2 lists the results of using the pupil dilation metric in some research studies, and the purpose of employing this measurement.

# Table 3.1: An overview of some eye-tracking metrics used in some recent studies

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper** | **Metrics** | **Focus** | **Findings** |
| (Wulff-Jensen et al., 2019) | * Fixation Counts * Saccade Length * Average Fixation Duration | To study the effects of using structural and textual code features on source code readability. | * The increase in the fixation duration metric is used as an indicator that missing structural/textual features negatively impact code readability. * The lack of structural features in the code resulted in a noticeable increase in the fixations count and saccade length. * Missing textual features on the code showed to increase in the gaze time. |
| (Soh et al., 2012) | * Average Fixation Duration * Normalized Rate of Relevant Fixations | The impact of expertise on UML class diagram comprehension | * Gaze time is used to measure the overall comprehension effort. * Experts show less fixation time and more accuracy on the task. |
| (Ahrens et al., 2019) | Heatmap | Studying the effect of using heatmap to visualize the attention distributed on the code areas to familiarize those who worked on the same source code afterward | The attention representation using heatmap provides helpful support to the developers, reducing their cognitive efforts while orienting them through a software maintenance task. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper** | **Metrics** | **Focus** | **Findings** |
| (Carniglia et al., 2012) | * Number of Fixation * Total Fixation Duration | Investigate the link between fixation features and the emotional aspect of a stimuli | * The emotional state when showing unpleasant pictures influences the subject’s gaze behaviors that are resulted in more fixations (efforts) and long gaze time (attention). * Animate stimuli require more attention, leading to a long duration than inanimate ones. |
| (İşbilir et al., 2019) | * Fixation Duration * Saccadic Amplitude * fNIRS | Assessing expertise using the measured cognitive workload through monitoring brain activity and tracking the subject’s eye movements. | * Gaze data and brain activation patterns can differentiate between experts and novices. * Experts show lower brain activity and short gaze time compared to novices. However, these measures increased significantly in the expert group after an unexpected error message showed to them. Therefore, experts are better at increasing their attentional focus to solve the problem than novices. |

**Table 3.2: An overview of using pupil size metric in some recent studies**

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper** | **Metrics** | **Focus** | **Findings** |
| (Couceiro, Barbosa, et al., 2019) | * Pupil Dimeter * HRV | Identifying code lines that have more mental effort required for comprehension | The pupil size variation combined with the heart rate variability provides helpful information to indicate the more mental effort devoted to specific code locations. |
| (Couceiro, Duarte, et al., 2019b) | Pupil Diameter | Can pupillography data be useful to assess the amount of cognitive effort spent on reading source code? | More mental effort is observed (pupil dilation) when reading source code that developers consider difficult and has high complexity. |
| (Behroozi et al., 2018) | * Pupil Size * Total Fixation Time * Regression Saccade | Measuring the stress rate in two interview settings: using a whiteboard vs. using papers during a problem- solving task | * Pupil size and fixation duration time are used to indicate and compare the cognitive efforts involved in two interview settings. * Using the whiteboard puts more pressure on interviewees, reflected in high cognitive load (pupil size increased) and less focus (less time spent solving the task). * Regression saccade (looking backward on the code lines) increased with whiteboard setting three times more than solving the task on paper. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Paper** | **Metrics** | **Focus** | **Findings** |
| (Kiefer et al., 2016) | Pupil Diameter | Studying pupil diameter response to solve map tasks as a measure for the level of cognitive load: low, medium, or high | Pupil diameter is a valuable indicator of perceived mental effort level based on the map complexity. |
| (Züger & Fritz, 2015) | * Pupil Size * Eye blinks * EEG | Predicting a good and bad moment of interruption in a real-time using psycho- physiological metrics | Analyzing mental state can be valuable index to measure the interruptibility of software developers automatically. |
| (Jimenez-Molina et al., 2018) | Pupil Dilation | Identifying multiple levels of cognitive effort while switching between web pages | * Measuring pupil dilation indicates four mental workload levels in short time windows. * Moving between web pages will reduce the mental effort compared to focusing on one web element. * A combination of pupil dilation metrics and other Psychophysiological sensors impact the accuracy of classifying mental workload levels involved in a web browsing task. |



# EXPERTS VERSUS NOVICES EYE MOVEMENTS

One way to assess expertise is to study programmer code reading behaviors at a finer level of granularity, namely the source code element level. We want to characterize the differences between experts and novices to determine how programmers (experts and novices) read source code at the construct level (e.g., loop statement, if statement, condition, block, etc.). Novices are known to cover more parts of the code than experts, but what exactly do they focus on mostly at the line, statement and term level when reading the programming code? Thus, this chapter presents the results relating to the research questions addressed earlier in section 1.2.

# Dataset and Task Used

The data set used in this analysis was collected in 2015 (Kevic et al., 2015). Kevic et al. conduct an eye tracking study on large source code in an open-source system using the Eclipse plugin iTrace (Shaffer et al., 2015; Guarnera et al., 2018). The eye tracking infrastructure iTrace automatically gives the exact source code element that is looked at with the line-level, by mapping the eye movement to source code elements even in the presence of scrolling and context switching. They used the Tobii X60 eye-tracker for the data collection which has 0.5 degrees of on-screen accuracy. Twenty-two programmers at two different expertise levels (experts and novices) participated in this study. The participants consisted of twelve professional programmers working in industry and ten computer science students. Each participant is asked to work and navigate through three

different change tasks (bug fix tasks). Twelve participants (nine professionals and three students) rated their experience in bug fixing as above average and the other ten rated their experience as average. They are given three bug reports from the JabRef repository, and they are asked to fix the bugs. The first bug is about a missing comma to separate the keywords and it requires the developers to traverse four classes to fix it, the second task is about fixing a failure to import big numbers, and the third task is about a failure to launch Acrobat on Win98 which requires a single method traversal to fix the bug. In this work, we use the eye tracking data related to the second task that required the programmers to read only one class to fix the bug.

# Data Cleaning and Transformation

This section describes and discusses the methods used in this investigation. The first part provides information about the data being analyzed, and the second part moves on to describe in more detail the pre-processing method applied to the data. Figure 4.1 presents an overview of the planning process we follow in the study to draw a conclusion.

# Source Code Data

To validate the comparison between experts and novices, we study the programmer reading behaviors of those who read the same source code. Thus, for this analysis we use the Java class file that contains the bug that programmers are looking at while finishing the task. We wanted to limit the scope of this analysis to just one class where the solution was found for the bug fix. Note that the participants did not know that the bug fix was scoped to just one class before beginning the study.

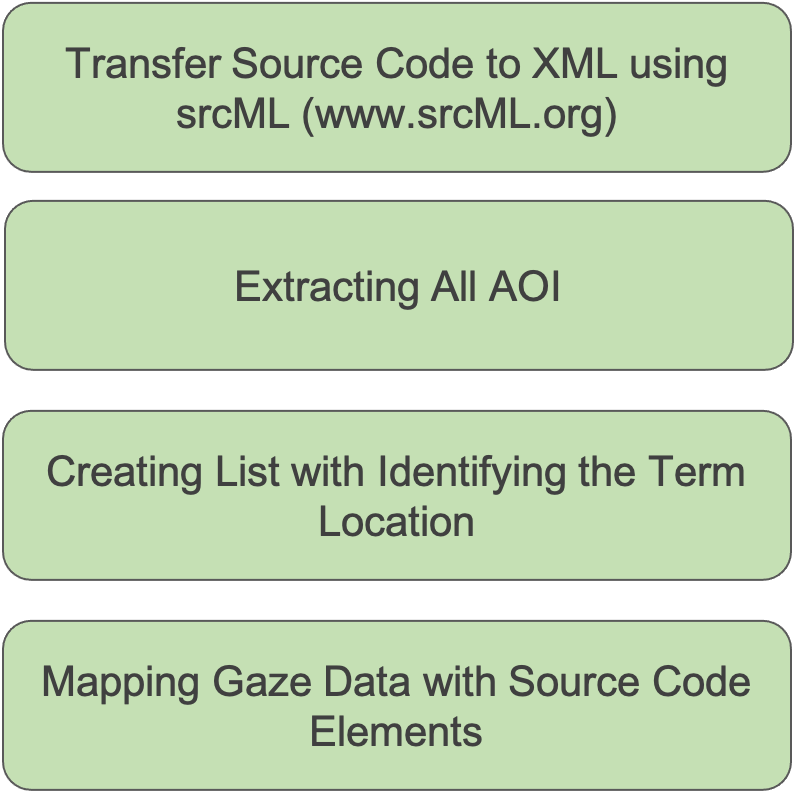
# Eye-Tracking Data

We analyze a total of 18 programmers eye movements data in this work, 10 experts and 8 novice programmers. We exclude four programmers from the analysis, as they did not look at the scope of the solution class where the bug is found. The eye tracking data includes information about the line and the column number where the programmers fixate on the code. Nevertheless, prior to undertaking the investigation, an IRB approval was obtained for the study

# Navigation within Source Code

For the scope of the solution in the Java class, we calculate the programmers’ eye movements over multiple parts of the Java code. The process of identifying the terms and statements that they look at is done by extracting the following source code element lists:

* Identifiers, including method and variable names
* Variables and method return types
* Names and operators in if, else, and while statements
* Operators
* Keywords
* Arguments
* Method signatures



# Figure 4.1: An overview of the analysis process

* + 1. **Identify Term Location and Mapping:**

To map the eye gaze onto the identifying source code elements, first we extract all the navigation areas that are to be studied from the code. We use srcML (Collard et al., 2011) to generate individual files for each navigation part which provides an XML representation of the code (Figure 4.2). Then convert these files to CSV lists. Each list (such as the operators list) includes information identifying the exact term, line number and the column number (the number of the beginning and the last column of the term).



# Figure 4.2: An example of the srcML generated code when extracting operators

In the mapping process, we map between the participants’ gaze data and the CSV lists to find the matching eye records. In particular, if the line of the eye tracking records matches the line in the list, and the column of the eye tracking records falls between the beginning and the ending column of the term, then we consider this element as viewed by the participant.

# Results and Discussion

This section provides the results obtained from the analysis relating to each of the research questions, along with the discussion associated with these findings.

# RQ1: Analysis at Term Level

This section describes the result of the RQ1: Considering programmer eye movements over Java source code, how do experts and novices compare when focusing on

method signatures, identifiers, types, operators, keywords, arguments, names, and types in if condition statement, else statement, and while statements?

First, we calculate the average of the source code coverage in each area of the navigation parts for the experts and novices. Then, we examine the significance of the difference between the two groups.

For the navigation areas investigated in the study, we found statistical evidence that novices read more source code elements than experts. In Figure 4.3, we present the results of comparing experts to novices in source code navigation.

# Identifiers:

With regards to identifiers coverage, our analysis shows that novices focus more on reading the names through the code navigation. Novices look at 23.7% of all identifiers in the code, while experts look at 9.5%.

# Signatures:

The result indicates that novices cover 54.7% of the method signatures in the code, while experts only cover 20.9% of the signatures.

# Keywords:

We found that novices look at 26.2% of the keyword list compared to experts, who read about 12% of all the keywords in the code.

# If-Then-Else Statements:

The results show that novices focus on more details while scanning the programming code than experts. In if-condition statements, we find that experts look at the names and operators less than novices. Novices look at 13.4% of the names in the if

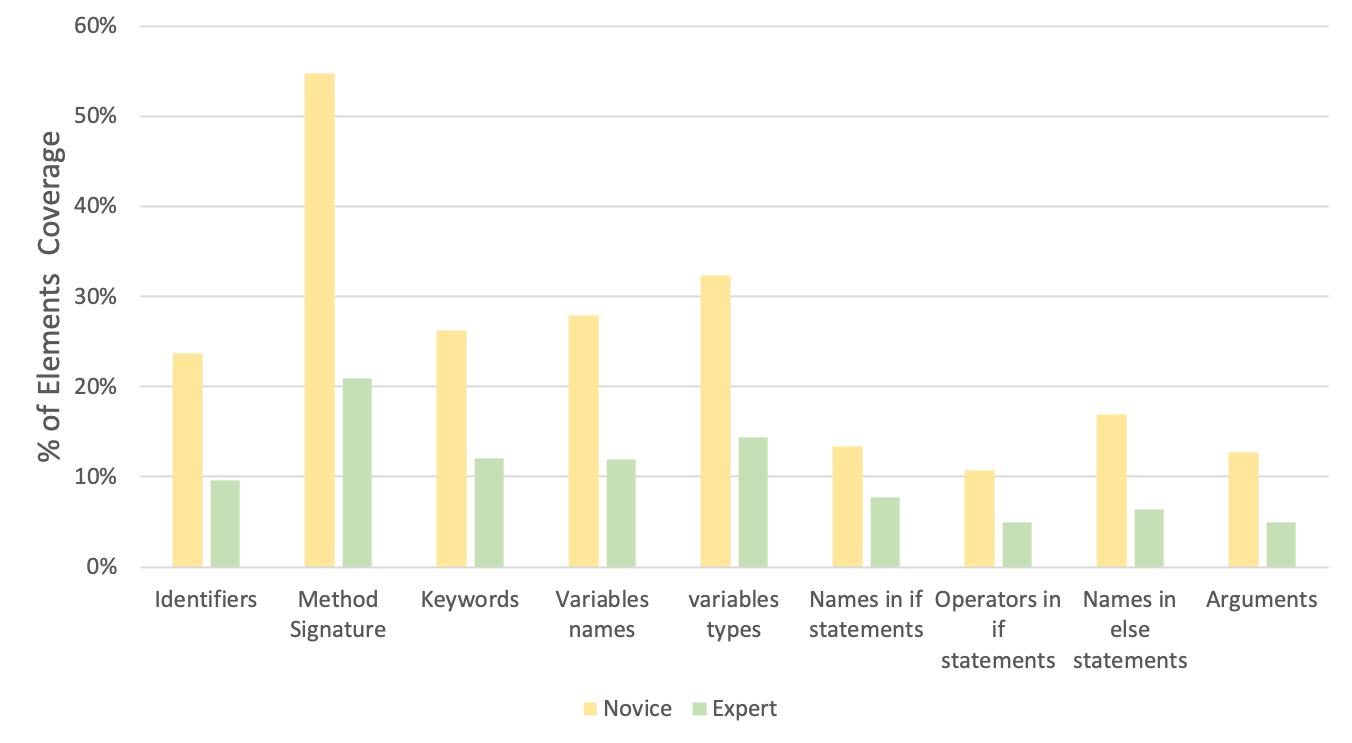
statements and about 11% of the operators. However, experts only look at 7.7% of the names and 4.9% of the operators in the if statements. Regarding the else statements, novices read about 17% of the names in the statements compared to the experts, who only read 6.4%. Novices look at 10.7% of the operators in the else statements, while experts only look at 1.4%.

# Variable Declarations:

During the scanning of variable declarations, Figure 4.3 clearly shows that novice programmers try to capture and read more in the declaration than the experts. About 28% of the names in variable declarations are looked at by the novices, while the experts only look at about 12% of all names. For variable types, novices cover 32.4%, while experts only cover 14.3%.

# Arguments:

Figure 4.3 shows that the difference between the two groups in focusing on the function arguments through scanning the source code is not large. However, experts look at 5%, whereas novices look at 11%.



# Figure 4.3: Comparing experts and novices eye movements in reading source code elements

* + - 1. **Expert vs. Novice**

After calculating the average of the source code coverage in each area of the navigation parts for the experts and novices, we examine the significance of the difference between the two groups.

We use the non-parametric Mann-Whitney test (Hollander et al., 2013) to compare the percentage of the element’s coverage between the programmers in the two experience levels (experts and novices). The results are presented in Table 4.1. We find the difference to be statistically significant between experts and novices in reading identifiers, keywords, method signatures, variable declarations, names in else statements, and finally, reading operators in else statements. We determined the significance based on the computed *Z* and *p* values, if *p* < 0.05 and the standard normal distribution |Z| > 1.96. Effect size is computed using Cohen’s d.

# Table 4.1: Mann-Whitney results for comparing novices and experts in navigation areas. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Source code elements** | ***U*** | ***P*** | ***Z*** | ***Cohen’s d*** |
| **Identifiers** | 65 | 0.027 \* | -2.217 | 1.149 |
| **Method signatures** | 13.5 | 0.02 \* | -2.329 | 1.122 |
| **Variable names** | 66 | 0.023 \* | -2.268 | 1.082 |
| **Variable types** | 64.5 | 0.033 \* | -2.138 | 1.071 |
| **Keywords** | 68 | 0.014 \* | -2.451 | 1.135 |
| **Arguments** | 54.5 | 0.212 | -1.247 | 0.960 |
| **Operators** | 59.5 | 0.091 | -1.689 | 1.114 |
| **Names in if statements** | 55 | 0.196 | -1.293 | 0.678 |
| **Operators in if statements** | 55 | 0.197 | -1.291 | 0.837 |
| **Names in else statements** | 65 | 0.023 \* | -2.273 | 1.294 |
| **Operators in else statements** | 61.5 | 0.024 \* | -2.256 | 1.239 |
| **Names in while statements** | 61.5 | 0.055 | -1.917 | 1.015 |
| **Operators in while statements** | 56.5 | 0.154 | -1.425 | 0.809 |

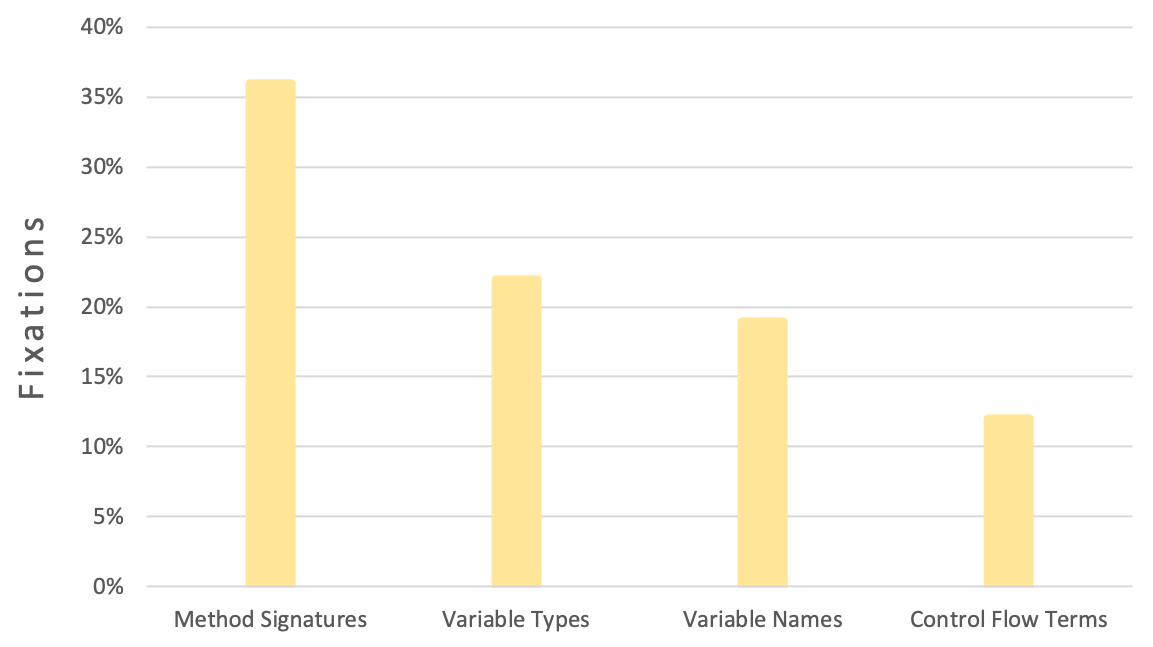
The results in Table 4.1 show that novices have a significantly higher average for reading identifiers in the source code. The difference is statistically significant: *U* = 65, *p*

= 0.027, *Z*= -2.217 with a large effect size of 1.149. The same holds true when comparing novice programmers with experts for reading method signatures; the difference between the two groups is statistically significant: *U* = 13.5, *p* = 0.02, *Z*= -2.329 Cohen’s d = 1.122. When comparing the keyword coverage, novices also have a significant higher average than experts in looking at keywords while scanning the code. The difference

between the two groups is statistically significant: *U* = 68, *p* = 0.014, *Z*= -2.451 with a large effect size of 1.135.

In reading variable declarations, we also notice that the average of covering declaration elements is significantly higher for novices. The results are similar when we compare the two averages of reading names in the condition else statements and the while statements. We find that the difference is statistically significant between experts and novices for covering names in else statements (*U* = 61.5, *p* = 0.023, *Z*= -2.273 and Cohen’s d = 1.294). The results also show that the difference is significant between experts and novices when looking at operators in else statements, *U* = 61.5, *p* = 0.024, *Z*= -2.256 with a large effect size of 1.239. However, the statistic result indicates that the difference between experts and novices for reading names in while statements is not significant: *U* = 61.5, *p* = 0.055, *Z*= -1.917 and Cohen’s d = 1.015.

Related to comparing the participants in reading arguments, operators in the code, and names and operators in if condition statements, we found that there is no statistically significant difference between experts and novices. This result indicates that both novices and experts tend to focus on fewer arguments and operators when they read the code. Experts look at 5% of the arguments and 7% of the operators, while novices look at 11% of the arguments and 12% of the operators. Results of all participants in reading each source code element including in this analysis are presented in APPENDIX A.



# Figure 4.4: Participants’ fixations distribution over source code areas

* + - 1. **Discussion**

These results imply that in this study, experts show better reading skills than novices. Experts focus on fewer parts of the code while scanning the visual stimuli. On the other hand, novices read more source code and focus on more details such as names and operators in condition statements than experts to solve the task. Thus, there is a specific difference between experts and novices while reading source code on a token level. Beyond assessing expertise while performing the task, these findings can also be applied to enhance the comprehension process, which can increase a programmer’s productivity. For example, identifying the lines and statements that have been viewed the most indicates difficulty to comprehend these lines which leads to providing appropriate help for guiding students through the learning process.

# RQ2: Total Gaze on Source Code Lines

Regarding the RQ2: Which parts of the code elements are looked at the most by the programmers (experts and novices)?

We find evidence that programmers focus on the method signatures the most while reading the code followed by variable types then the variable names. On average, programmers cover 36% of method signatures, while they view variable declarations by looking at 22% of the variable types and 19% of the names. However, control flow statements receive the least amount of focus by programmers, who look at no more than 12% of the names in while statements. Figure 4.4 illustrates fixation percentages over multiple source code areas.

# Discussion

In terms of the navigation area that is most visited when programmers review the code, the results show method signatures as the most viewed area. This conclusion confirms what has been found in prior eye tracking studies that investigated programmers’ code reading behaviors (Rodeghero et al., 2014; Begel & Vrzakova, 2018). In particular, during the scanning process programmers tend to focus more on the signatures before they decided to move to the body to look at more code elements. In contrast, Abid et al. claims that programmers read method signatures less than the other terms (calls and control flow) (Abid, Sharif, et al., 2019). They conclude that programmers look at the methods’ body more than the signatures. Compared to our findings, one explanation for this difference is that in Abid’s work, programmers read the source code to provide a written summary for the methods, which requires them to have a close look at the method's body. Also, the use

of multiple long and complex methods from different domains (as in the Abid study) also caused this difference since it reflects on the programmers’ comprehension process in a different setting. Therefore, in their study, for summarization tasks, programmers read the method's body more to write a correct summary.

# Conclusion

The first step toward achieving our research objective is to study the impact of expertise on source code reading behaviors. This has been accomplished by analyzing an existing eye tracking data set to study programmers’ behaviors and strategies during source code reading in the context of a bug fix. We use multiple navigation areas from the source code as measures to compare participants with two types of programming experience levels (experts and novices). Our findings from the analysis of applying the measures on the eye movement data show that experts and novices are different when reading source code. For all source code elements examined in the study, we find that the novices look at more elements than expert programmers. Also, the novices tend to read more details within the code than the experts. This has shown that determining differences in reading behavior between experts and novices by investigating eye movement on source code is a practical approach for expertise assessment.



# EXPERTISE VIA EYE- TRACKING MEASURES

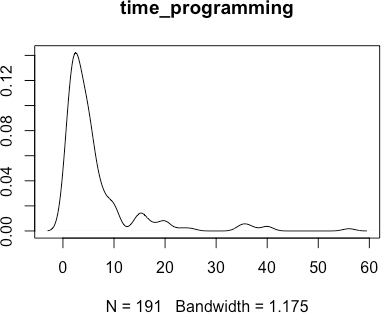
The objective of this study is to provide a reliable approach to characterize developers’ expertise level as expert/novice via findings from eye movement data. This study also aims to make use of the eye movement related metrics to assess participants visual efforts and their cognitive workload during a program comprehension task with respect to their expertise.

# Study Design and Procedure

This section introduces the eye-tracking metrics that we use in this study to assess expertise during comprehension program activities. Next, the process of cleaning and transforming the dataset is given, followed by the participants’ descriptions relative to their expertise evaluation.

# Dataset

This analysis used the EMIP dataset known as Distributed Collection of Eye Movement Data in Programming (Bednarik et al., 2020). This dataset is part of the EMIP international workshop which started in 2013. Due to the lack of a large eye movement dataset availability, multiple labs from different countries contributed on an eye-tracking experiment to enrich the research field with a large gaze dataset. They used the eye tracker SMI RED 250 mobile to record the data which were shipped to the participants with all the instructions about running the experiment. To date, the published EMIP dataset has the eye gaze data of 216 participants who conducted the program comprehension experiment with

two visual stimuli which are available in three different programming languages. Their years of programming experience ranged between zero and 56. The two programming tasks are called ‘Rectangle’ and ‘Vehicle’, each of which includes a class that matches the name of the task. Subjects were asked to choose a programming language between Java, Python, or Scala with which to start the experiment. After they had finished reading the code, participants were asked to solve some comprehension questions to assess their understandability of the code. The majority of the subjects used Java program (207 participants), 5 chose Python, and four worked with the Scala. We used in this analysis the eye-tracking data for participants who worked with Java stimulus. Figure 5.1 shows the distribution of the participants’ years of experience in the study. Most of the subjects who participated in the experiment fall within 1 to 10 programming experience years.

# Figure 5.1: Participant's expertise distribution in EMIP for Java code

# Measures

There is a broad range of eye tracking measurements that have been explored and used by researchers in previous software engineering eye tracking studies (Sharafi, Shaffer,

et al., 2015; Sharafi, Soh, et al., 2015). We choose the metrics from three main categories in this study: fixation-related metrics, saccade-related metrics, and pupil dilation metrics. However, we first aim to study the relationship between multiple types of developer expertise metrics and their eye movement metrics. This in turn will allow for finding the best metrics to use when explaining the differences in levels of expertise by using eye- related metrics.

# Visual Effort Assessment

With respect to measure the visual effort, we use five eye-related metrics to analyze the data and assess the participants’ visual attention efforts including:

# Fixation Metrics

Fixation happens when the eye is focused and stabilized on one location among the source code elements for a short time. The following fixations related metrics are included in this study:

# Number of Fixations

The total number of fixations that the subject used over the programming tasks (Rectangle.java and Vehicle.java). This metric is calculated for both tasks that the participant used after cleaning the data and removing all invalid eye records. Many of the previous eye-tracking studies use fixation metrics to detect and measure either the search efficiency (Goldberg & Kotval, 1999; Jacob & Karn, 2003) or to find the most focused areas of interest (Crosby & Stelovsky, 1990; Uwano et al., 2006). Overall, a higher number

of fixations indicates a lower efficiency to search for relevant information in an AOI or stimuli. This leads to more visual effort required to complete the task.

# Line Coverage

This metric represents the percentage of lines the participant looked at from the program lines. Eye gazes are mapped to the corresponding source code line number by using the information provided with the EMIP dataset (Bednarik et al., 2020). The stimulus information included with the dataset shows the lines coordinates for each program. The AOI model proposed by Deitelhoff et al., is used in the mapping process between the eye records and the code lines; this allows the researchers to capturing more AOI (code lines in our study) that can be used mainly in comprehending the code (Deitelhoff et al., 2019). This AOI model approach utilizes the gap between the code lines due to the possibility of having fixations that fall in that space. In this study, we note that applying this method of no vertical margins between code lines increases the number of fixations mapped to the accurate line number. However, it should be mentioned that our intent when calculating the line coverage metric is to capture more AOI transitions in the mapping process, and this requirement is satisfied and proven by Deitelhoff et al. (Deitelhoff et al., 2019).

# Sum Fixation Duration

This represents the sum of all intervals between two fixations. Thus, we calculate the fixation duration that subjects use at each eye record by taking the difference between the timestamp at one fixation (Fi) and the subsequent fixation (Fi+1). However, fixation duration is used to get insight into developer performance and their overall visual attention that require them to work on the task (Bednarik, 2012; Soh et al., 2012). A longer fixation

duration means that the participants spend more time understanding and analyzing the task (Cepeda Porras & Guéhéneuc, 2010) due to code complexity (Busjahn et al., 2014) or defect difficulty (Cagiltay et al., 2013), and reference to the more cognitive effort is needed to explore the whole given tasks (Sharafi et al., 2013). Although fixation counts and gaze time represent developers’ visual efforts to perform programming tasks, they are not correlated (Sharafi et al., 2020).

# Saccade Metrics

A saccade happens between two consecutive fixations. It is a form of navigation behavior. No processing happens during saccades.

# Saccade Length

This study defines the saccade length as the sum of the Euclidian distance between consecutive fixations divided by the number of saccades (Sharafi, Shaffer, et al., 2015).

# Saccade Duration

In this study, saccade duration represents the sum of the duration of all saccades divided by the total number of saccades.

# Cognitive Effort Assessment

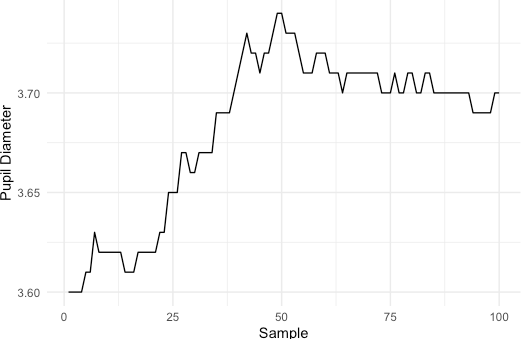
This work also uses the changes in the pupillary response of expert/novice developers as an indicator for the underlying cognitive efforts they performed. Relatively, in order to provide a valid dilation analysis and to perform a fair comparison between developers, this study includes developers who solve tasks in the same language (Java).

# Pupil Size Metrics

When the pupil gets dilated, it allows for more light in the eye. However, it happens in different conditions such as in low light situations or as a result of changes in the cognitive efforts of the person (Granholm & Steinhauer, 2004; Goldinger & Papesh, 2012) (Eckstein et al., 2017). Thus, pupil dilation can provide a sensitive index to reflect the cognitive challenges when working on a difficult task (Kahneman & Beatty, 1966; Klingner et al., 2011; Fritz et al., 2014). Therefore, instead of using the exact pupil diameter this study’s analysis use the pupil dilations to measure the differences in memory load between the two groups (experts/novices) (Klingner, 2010) as follows:

# Measure Pupil Dilation

This represents the increase in the pupil’s diameter size measured in mm. Since pupil size can increase up to 0.5 mm as a response to an increase of cognitive load, here, we consider four maximum values in our study’s analysis (0.1, 0.2, 0.3, 0.4) to capture small changes in pupil size increasing over the baseline. We exclude the changes of 0.5mm in the pupil size analysis because only a few developers have recorded dilation up to this point, especially with experts (only 13 experts). We use a baseline interval of 60 ms (15 samples) from starting the task (reading the code) and take into account the trial with which the subject starts. Furthermore, we detect the baseline as the average of the developer’s pupil size in the first 60 ms after starting fixation on the target (code). For each developer, the relative pupil dilations are obtained by subtracting the baseline value from each pupil diameter value in the trial. Then, we calculate the percentage of the fixations when a subject’s pupil gets dilated above their baseline in each tested point. Figure 5.2 and Figure

5.3 show the timeline of pupil diameter (100 samples) for both novice (N62) and expert (E1) while reading the first trial.

(mm)

# Figure 5.2: The timeline of the pupil diameter for novice (N 62) at the beginning of reading during the first trial (Rectangle. Java), where the baseline averaged to 3.61 mm from the first 15 samples.

(mm)

**Figure 5.3: The timeline of the pupil diameter for expert (E1) at the beginning of reading during the first trial (Vehicle. Java), where the baseline averaged to 2.57 mm from the first 15 samples.**

# Evaluate Expertise

There are 207 participants we consider in this study. Their years of programming experience ranged between zero and 56. Based on the fact that programming skill increases with the years of experience in the field (Yu et al., 2019), and based on the findings of the first research question, we classify experts and novices within the experimental group by using their years of programming experience. The median value of the programming experience information provided by the participants (4 years) is used. Excluding those with none (zero) experience, which makes 16 participants out of the total, there are 101 experts with 4+ years of experience, and 90 novices with 0.5 to 4 years of programming experience. Figure 5.1shows the distribution of the participants’ years of experience in the study. Most of the subjects who participated in the experiment fall within 1 to 10 programming experience years.

# Data Cleaning and Transformation

In this step, we prepare the data for the analysis and filter the collected eye records. First, we extract the eye records related to each stimulus program (Rectangle.java and Vehicle.java, with 18 - 22 LOC) from the raw data for every subject and save the result in CSV files. Then we check the data for validity, including removing all records when the eye pupil position is zero. In total, we include in this study’s analysis 207 participants who worked in two different comprehension tasks in Java. To validate the comparison and make it easier to compare between participants’ eye measurements, this analysis uses a common scale for all the data without changing the range between the values. So, we utilize the min- max normalization to transform all the data records to the same scale by subtracting the

minimum from each value, then dividing the result by the difference between the max and the min value (max-min scaling). Thus, the minimum value transformed to zero, and the maximum value transformed to 1. The results section continues with description of the other types of filters and cleaning that are applied to calculate the measurements.

# Results and Discussion

We combine the two trials’ results (Vehicle and Rectangle) dataset for all 207 participants (novices and experts) who read Java source code, creating a total of 414 data points. To compare the studied eye-tracking measurements of the experts and novices and to find any significant differences between them, we provide the results to our research questions.

# Correlation Between Expertise and Eye Tracking Metrics

The answer to this question shows the correlation between expertise and eye-related metrics. To overcome any errors by dividing the participants into two different groups of expertise we ignore the variance between their years of programming. This in turn will allow for finding the best metrics to use when explaining the differences in levels of expertise by using eye-related metrics.

Therefore, in this section we aim to answer the following research question:

* + - * RQ1: Which is the best representative measurement for estimating expertise that shows a best connection with eye-tracking metrics?

We define a set of independent metrics that have been utilized in previous studies to assess and measure developer’s expertise. All metadata related to subjects is provided

with the EMIP dataset in a separate file (Bednarik et al., 2020). We use the following variables to represent expertise:

# Time Programming:

This metric is measured as years of practicing coding, which represents the subjects’ programming experience in general. In this analyzed dataset (EMIP), the distribution of the subjects’ years of programming is shown in Figure 5.1. The years of experience ranged between 0 and 56.

# Time Programming in Experiment Language:

This measured as years of programming in the experiment language Java. Relating to the analyzed dataset, 216 participants chose to use Java programs to conduct the study. The subjects' years of experience range between 0 to 30 years of using the Java programming language.

# Self-Evaluation in Programming:

This variable shows the developers’ self-evaluations of their programming experience. Subjects chose between none, low, medium, or high. However, the metric receives one of the values: 0 if the subject chooses none for programming experience, 1 for low, 2 for medium, and 3 for a developer with high expertise.

# Self-Evaluation in Experiment Language:

This metric aims to show how developers evaluate their expertise in the Java programming language. It gets a value of 0, 1, 2, or 3 based on participants’ entries (none, low, medium, or high).

# Rectangle Task Performance and Vehicle Task Performance:

This binary metrics take a value of 1 if the developer solved the task correctly and a value of 0 if the developer doesn't choose the correct answer from the comprehension questions after reading the program.

To investigate RQ1, we determine the Spearman rank correlation between the extracted eye movements parameters and the expertise metrics. Because of the highly skewed distribution of the data, we adopt the non-parametric Spearman rank correlation, which does not assume either normality distribution or linearity association between values, and robust to outliers (Schober et al., 2018). For each pair of metrics, we compute the Spearman rank correlation coefficient to evaluate the strength of the relationship between eye measurements and expertise variables. Figure 5.4 contains a color-coded correlation matrix that shows the correlation results between expertise and gaze behavior measurements. We observe that time programming is the expertise variable that correlates with eye-related metrics with the highest correlation coefficient. Typically, a correlation coefficient with the amount of time programming shows small to medium results | *rs* | >

0.1 and | *rs* | ≲ 0.3 in all eye-related metrics but saccade duration and line coverage (Vehicle task), based on the Cohen correlation coefficient interpretation.

We obtain a negative correlation between the fixation-related metrics, including fixations count and total duration and the time of programming representing expertise level. Also, this correlation is statistically significant with *p* < 0.0001. It is the same with pupil dilation parameters, as they are negatively correlated with years of programming and statistically significant with *p* < 0.0001. However, the time of programming correlates

positively with saccade metrics, and it is statistically significant with saccade length, and not with saccade duration.

We also noticed that the correlation with pupil dilation increased with the pupil diameter. The metric of pupil size increased up to 0.4 mm; this has the highest correlation results compared to other pupil-related metrics with time programming by *rs* ≈ 0.3. However, we can say that the years of programming experience, in general, is the most accurate metric to use with eye-tracking parameters to estimate expertise in this study.

The same result is derived in the original dataset collected paper (Bednarik et al., 2020). However, adding this study’s correlation analysis further in our work strengthens the case for choosing the years of programming experience to represent expertise statistically.

# Discussion

Comparing correlation results across all expertise variables with eye movements parameters shows that using the years of programming experience is the best variable choice to explain overall expertise in this analysis. However, this connection’s results reveal insights about the importance of focusing on teaching the programming logic regardless of the language’s syntax for beginner programmers. Almost all of the parameters that captured the participants’ gaze reading behaviors correlated with programming time higher than that of the other expertise metrics. Thus, we can conclude that expertise represented by the years of programming experience can play an important role on the source code reading behaviors in this experiment. However, this result does not exclude the fact that other expertise metrics could also have an impact on code comprehension,

which reflects on developers reading strategies. Nevertheless, in this experiment measuring expertise by the time of programming was the best fit to interpret eye parameters successfully. For this reason, replication experiments with different settings are important to generalize the results, such as assigning participants in multiple groups with different programming languages for each.

# RQ1: Summary

Overall, the correlations of developers’ years of programming experience indicate a small to a moderate relationship with each eye-tracking metric (except saccade duration and line coverage in Vehicle task) which performs the best association across approximately all expertise metrics. Thus, this result suggests that using the years of programming is the most accurate metric to represent expertise in this study. In addition, the study’s findings provide empirical evidence that the more time spent practicing programming regardless of the language that a developer is particularly interested in, the more noticeable increase the probability of reading programming code efficiently, along with decreasing the cognitive workload, and improving the program comprehension process.

Because the existence of a correlation does not necessarily mean causation, further investigation is needed to examine the influence of expertise in a particular language or domain that would affect the developer performance in other programming languages.

Vehicle.Task.Performance

**−0.2007**

**0.1374**

**0.0623**

**−0.2902**

**−0.2539**

**−0.2431**

**−0.2184**

**−0.2008**

**0.006**

**−0.1599**

**−0.0897**

**0.1295**

**0.0282**

**−0.1082**

**−0.0731**

**−0.0378**

**−0.0165**

**−0.1235**

**−0.0721**

**0.0779**

**0.0724**

**−0.0125**

**−0.0193**

**−0.0346**

**−0.0079**

**−0.0174**

**−0.0522**

**0.0844**

**0.0743**

**0.0088**

**0.0695**

**0.1688**

**0.0345**

**0.0235**

**0.0637**

**0.0102**

**−0.0334**

**0.094**

Rectangle.Task.Performance

Spearman rank Correlation

1.0



|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Self.Eval.Experiment.Language | **0.0128** | **−0.0324** | **−0.0257** | **0.0922** | **0.0818** | **0.0368** | **−0.0191** | **0.0795** | **0.0604** | **0.0086** |
| Self.Eval.Programming | **−0.2267** | **−0.0955** | **−0.0698** | **−0.1121** | **−0.1196** | **−0.1407** | **−0.1804** | **0.0166** | **0.0704** | **−0.0909** |

0.5

0.0

−0.5

−1.0

Time.Experiment.Language

Time.Programming

# Figure 5.4: Correlation matrix of the correlation coefficient result between eye-movement measurements and expertise

70

# Metrics Between Experts and Novices

In this section, we answer the following research question:

* + - * RQ2: Using eye movement measurements: fixation counts, total fixation duration, code lines coverage, saccade length and saccade duration, to what degree do experts differ from novices?

To address this research question, we calculate the metrics described in section (5.1.2) to empirically investigate the usefulness of eye movement parameters in the context of expertise assessment. The result of the comparative study between novice and expert developers in the calculated metrics is presented in Figure 5.5. Novices tend to have noticeably higher reading parameters with fixation-related metrics, whereas experts record a higher average on their saccade-related metrics than that of the novice developers. However, the line coverage measurement varies with the task performed. Table 5.1 shows the average of the actual data and the standard deviation in fixation and saccade metrics.

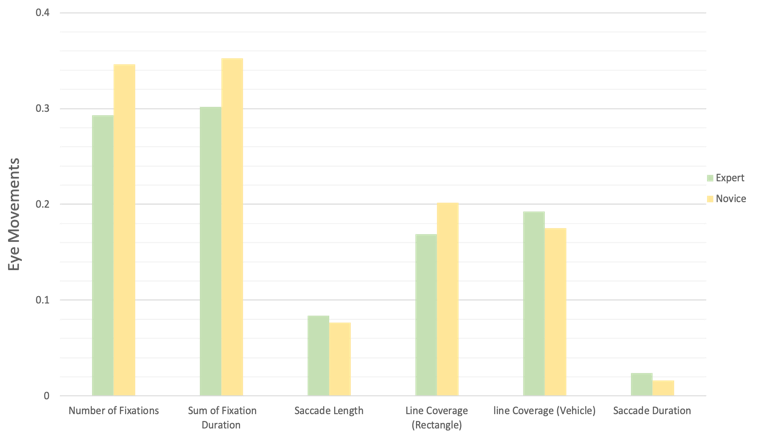
To determine the significance, we test these differences between experts and novices using the non-parametric Mann-Whitney test (Hollander et al., 2013) based on the normalized values. We apply the non-parametric Mann-Whitney test because the data is not normally distributed. From Table 5.2, the results show that the difference between experts and novices is statistically significant in fixation counts, total fixation duration, saccade length, and line coverage in the Rectangle program. In the Vehicle trial, experts cover more lines in the program than novices, but the difference is not significant. Thus, less experienced subjects exhibit more fixations on the code with a longer gaze time

duration and a shorter saccade length; which means they do short skip over the program lines more often than skilled subjects.

# Table 5.1: The result of experts and novices on eye tracking measures

|  |  |  |  |
| --- | --- | --- | --- |
| **Eye Movement Feature** | **Level of expertise** | **Average of actual data** | **SD** |
| **Number of Fixations** | Experts | 12500.58 | 7516.89 |
| Novices | 14656.87 | 7046.74 |
| **Sum of Fixation Duration** | Experts | 56474.88 ms | 30941.87 |
| Novices | 65341.34 ms | 30347.74 |
| **Saccade Length** | Experts | 4.53° | 4.58 |
| Novices | 4.17 ° | 5.22 |
| **Saccade Duration** | Experts | 5.9 ms | 6.65 |
| Novices | 5.1 ms | 7.03 |

(0 ~ 0.4)

**Figure 5.5: Experts vs. Novices in fixation-related metrics and saccade-related metric**

# Number of Fixations Result

We examine the fixations distributions of both experts and novices over the source code. We find statistical evidence that novices have a larger number of fixations when comprehending the task. This result explains the relative attention that novice developers devote to understanding the source code compared to the experts. On a scaled average, experts use 0.29 fixations to comprehend the source code, while novices need 0.07 more fixations to read the code. In the actual reading shown in Table 5.1, experts average 12500.58 gaze visits in reading both trials, while novices have an average of 14656.87 visits. Based on the computed probability and *Z* presented in Table 5.2, we show that the difference between the two groups in the devoted fixations is significant. The results of the

non-parametric Mann-Whitney test for the number of gaze code visit is (*Z* = -3.748, *p* = 0.00018) with a medium effect size of *d* = 0.312. This result indicates that expertise influences the total number of fixations required to understand the programs. This reading behavior is also reported in the writeup of the EMIP dataset (Bednarik et al., 2020), but here we shed light on the degree of these differences between experts and novices when testing the significance of this result.

# Total Fixation Duration Result

We find evidence that novices have a statistically significant higher percentage of gaze time views of the source code than that of experts. Using actual duration in Table 5.1, experts averaged 56474.88 ms of their gaze time viewing the programs while novices spent 65341.34 ms of the duration reading and understanding the programs. We determined the significance based on the computed *Z* and *p* values (as shown in Table 5.2). The results show that the average novices’ fixation time is significantly higher than experts: *U* = 14190, *p* = 0.00021, *Z* = -3.703 with a medium effect size of *d* ≈ 0.31. This indicates that expert developers with high programming experience are more efficient in using their prior knowledge, thus comprehending the task faster than novices. This observation aligns with the results visualized in the EMIP dataset paper on the influence of expertise (Bednarik et al., 2020). However, the findings of this study emerge from statistical analysis to provide further explanation about expertise effects on fixation duration measure.

# Table 5.2: Mann-Whitney test result of experts and novices in fixation-related metrics and saccade-related metrics. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Eye Movement**  **Feature** | ***U*** | ***p*** | ***Z*** | ***d*** |
| **Number of Fixations** | 14142 | 0.00018\* | -3.748 | 0.312 |
| **Sum of Fixation**  **Duration** | 14190 | 0.00021\* | -3.703 | 0.309 |
| **Saccade Length** | 20808 | 0.01474\* | -2.439 | 0.077 |
| **Saccade Duration** | 19286 | 0.3046 | -1.027 | 0.089 |
| **Line Coverage (Rectangle)** | 3791.5 | 0.04353\* | -2.019 | 0.35 |
| **Line Coverage (Vehicle)** | 4914.5 | 0.3266 | -0.981 | 0.159 |

# Saccade Length Result

It is observed that experts tend to make a longer saccade while reading the programs than the novices. The results of the average saccade length for all source codes show that experts had an average length of 4.53° while the novices’ average saccade length is 4.17° (Table 5.1). When testing this difference with the Mann-Whitney test (Table 5.2), we find that the average experts’ saccade is significantly higher than that of novices’, with *p* = 0.01474, *Z* = -2.439, *U* = 20808, and a small Cohen effect size *d* ≈ 0.1. This result was also observed previously by findings in Busjahn et al. (Busjahn et al., 2015): that experts are

better when deciding where to look in the code and can therefore make larger skip to find important parts to understand the program.

# Saccade Duration Result

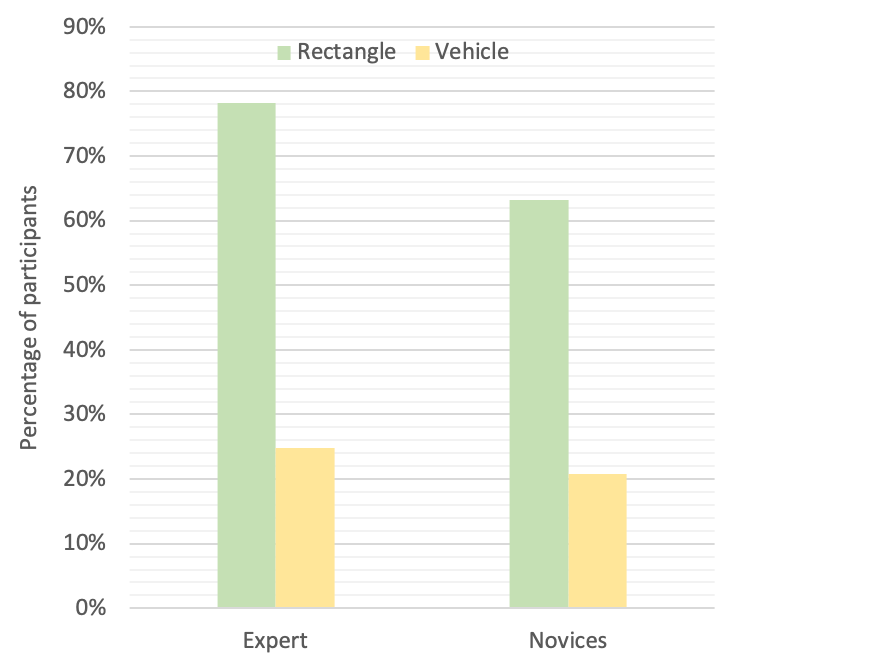
Related to the saccade duration, we find in this analysis that the saccade duration for the experts lasted between 4 ms and 50.75 ms for each saccade, whereas the novices spent from 4 ms to 83.816 ms in each saccade. On average, experts recorded longer saccade durations than the novices by using on average 6 ms on saccades per task, while novices took an average of 5.1 ms (Table 5.2). However, this difference is not significant between experts and novices for saccade duration (*p* = 0.3046, *Z* = -1.027) with a small effect size *d* ≈ 0.1 (Table 5.2).

# Discussion

Although the result does not show any significant difference between experts and novices in saccade duration, experts used a higher average of saccade duration than novices, thus showing that experts tend to be more sufficient when reading the source code. We conclude that although experts used a significantly lower fixation count than the novices, they spent enough time at each saccade to acquire the information they need to comprehend the task. Thus, experts in this experiment are better at reading the task with less fixation and time but with more focus. More filtering and combining for fixations are important to apply in order to improve this result.

# Line Coverage Result

Reading the Rectangle task shows that novices recorded a higher average of line coverage than that of the experts. From mapping the fixation counts to the line number coordinates using the no vertical margins method (Deitelhoff et al., 2019), we find that novices read 20% of the Rectangle class lines, while experts covered about 17% of the code lines. This difference between experts and novices in their code reading coverage is significant with a medium effect size (*p* = 0.04353, *Z* = -2.019, *d* = 0.35). However, with the second task the results show that the average of reading code lines in the expert group is higher than that of the novice group, with 19% for experts and 17% for novices. Nevertheless, this result is not significant for this task (*p* = 0.3266, *Z* = -0.981, *d* = 0.159).



# Figure 5.6: Participant correctness level for Java code in Rectangle and Vehicle tasks

**Discussion**

with studying reading behavior of developers, researchers have determined that experts and novices read code differently when it comes to code word coverage (Busjahn et al., 2015), or even in more fine-grained level reading source code elements (Aljehane et al., 2021; Madi et al., 2021). Thus, this study’s result in analyzing reading behavior at line level for task (Rectangle) matches those found in previous findings.

However, the developers’ performance of the Vehicle task may explain the different reading strategies by developers. In the Vehicle task, only 47 developers solved the task and comprehended it correctly, while 160 did not answer the comprehension questions correctly. Figure 5.6 shows the participants correctness results for those who chose Java for each task with expertise consideration. According to the authors of the EMIP dataset experiment, subjects understood the main idea of the program but missed the tricky part of the effect on the Vehicle speed by passing a negative number to the acceleration method (Bednarik et al., 2020).

# RQ2: Summary

To summarize, we find strong statistical evidence concerning the usefulness of eye tracking measurements in the context of assessing expertise, confirming the findings of previous studies. Participants with lower programming experience have devoted significantly higher fixations with longer durations to comprehend the code than the expert developers. Moreover, we observe that experts tend to make longer saccade lengths when moving between the code elements of longer saccade durations than the novices.

# Cognitive Load and Pupillary Response

This section answers the following research question:

* + - * RQ3: To determine the differences between the cognitive load effort of experts and novices, can a developer’s pupillary response contribute to assessing expertise?

Addressing this research question aims to detect evidence about an expertise impact in the cognitive process of subjects. Therefore, for each developer, we study their pupillary responses to the reading and understanding process of the two visual stimuli, after which provide an interpretation of the differences between the experts/novices groups.

We apply outliers filtering on the dilation data, as it is likely to see errors caused by blinking, for example (Sirois & Brisson, 2014). In the details, we calculate the boxplot over the fixations with dilation up to 0.1 mm. As a result, 6 participants from the analysis whose pupils dilated up the baseline value to zero, 5 of whom were wearing glasses, were excluded. Also, in the process of removing the outlier values, we have noticed that outliers in dilation include the data of 23 participants for either the Rectangle task or the Vehicle task. Out of those participants, the majority of the cases (17 participants out of 23), the dilation data of the first trial was considered as an outlier value. The result, in fact, shows that the pupil dilation data in the second task has reasonable differences relative to the baseline pupil diameter and gives better statistic results. Therefore, as a result, 379 such eye movement data points were included in this analysis (out of all 414 data points).

Figure 5.7 presents the results of the developers’ cognitive load analysis using a pupil dilation indicator. We compared the distributions of the increase in pupil size above

the baseline; for all the tested peak values, novices show a significant higher average of fixations with dilated pupils than the experts. In detail, for a pupil size increase by 0.1 mm, the results show that novices experience a higher average of fixations with dilated pupils than the experts. Experts have 7.73% of their fixations where their pupil sizes tend to be dilated up to 0.1 mm, while novices recorded an increase in the pupillary response with 12.17% of the total fixations. The same thing is discovered when comparing the average of fixations with dilated pupils up to 0.2 mm: we notice that novices experience pupil dilation more often (7.9 % of their fixations) than those of experts (2.3 % of their fixations). In the case of the differences at 0.3 mm, the analysis shows that less than 1% of experts’ fixations have an increase in pupil diameter size up to 0.3 mm, whereas novices have an average of 3.2 % of these same fixations. The results are similar with the differences at 0.4 mm; we find that on average, novices’ fixations with a pupil dilation up to 0.4 mm is 1.2 % of their total gaze distribution, compared to experts who recorded less

than 1% of fixations (0.34%) with their pupil dilated up to the same value.

Testing these differences between experts and novices in the pupillary response using the Mann-Whitney test shows that the difference is statistically significant in all tested maximum values of pupil dilation above the chosen baseline. The analysis shows that the significance increased with the pupil size from 0.1mm to 0.4mm, but the tested effect size decreased (Table 5.3). As shown in Table 5.3, based on the calculated *p* and *Z* values, we reject the null-hypothesis that stated there is no difference between experts and novices in their pupillary response, which represents the cognitive workload. When data of both trials are combined, the results of the Mann Whitney test for the increase in pupil

diameter up to 0.1 mm and 0.2 mm is (*U* = 12533, *p* = 0.00934, *Z* = -2.599 with a large Cohen’s d effect size of *d* = 0.519) and (*U* = 11500, *p* < 0.001, *Z* = -3.882 with a large Cohen’s d effect size of *d* = 0.52), respectively. Similarly, we found an evidence that the difference between experts and novices in their pupil dilation up to 0.3 and 0.4 mm is statistically significant with a medium effect size (*U* = 11813, *p* < 0.001, *Z* = -3.714, Cohen’s *d* = 0.447, for pupil size increasing up to 0.3 mm) and (*U* = 11988, *p* < 0.0001, *Z*

= -4.059, Cohen’s *d* = 0.37, for pupil size increasing up to 0.4 mm).

# Discussion

We found a prove that pupil size corresponds with expertise level. Novices show a significantly higher average of fixations with an increase in their pupil sizes relative to the baseline. We can therefore conclude that less experienced developers perform a higher workload effort and focused attention while processing the comprehension tasks which support the hypothesis that reported by Kontogiorgos and Manikas in (Kontogiorgos & Manikas, 2015). This may suggest that novices see the task as difficult which explains the increase in their pupil size. This was pointed out by Freitz et al. (Fritz et al., 2014) who find that changes in pupil size is distinguishable metric to decide and predict task difficulty. Also, the same result of studying difficulty connections to pupil diameter has been previously proved by Hess and Polt that the more difficult the problem gets, the more the participants’ pupils dilated (Hess & Polt, 1964). To the best of our knowledge, this current work is the first in the field to study the cognitive model of developers while performing a comprehension task and find the level of comparison between experts and novices in this

metric.

(0 ~ 0.15)

# Figure 5.7: Experts vs. novices in pupil-related metrics

**Table 5.3: Mann-Whitney test result of experts and novices in pupil-related metrics.**

# Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pupil Dilation** | ***U*** | ***p*** | ***Z*** | ***d*** |
| **Pupil Size Increase 0.1** | 12533 | 0.00934\* | - 2.599 | 0.519 |
| **Pupil Size Increase 0.2** | 11500 | 0.000104\* | - 3.882 | 0.52 |
| **Pupil Size Increase 0.3** | 11813 | 0.000204\* | - 3.714 | 0.447 |
| **Pupil Size Increase 0.4** | 11988 | 0.000049\* | - 4.059 | 0.37 |

# RQ3: Summary

This study aims to characterize these changes in ocular responses, such as measuring pupil dilations. Within all tested pupil diameter values, the analysis identifies significant differences between experts and novices in the pupillary response; these range between medium and large effect sizes. This result would suggest that since the pupillary response is a promising measure of mental effort, it can be used successfully along with other eye-tracking metrics to find developer differences in terms of expertise levels.

* 1. **Implications**

This study analyzes developers’ eye movements while comprehending programming tasks and introduces an approach to assess the participants’ expertise levels by using several eye measurements and change in the mental workload. We use pupil size as a measure of the mental load. Many studies show a connection between changes in pupillary response and the cognitive process required to perform a task (i.e., perceived task difficulty level). The work presented here provides further evidence showing the role of expertise on the developers’ cognitive efforts. Additionally, the result of the analysis provides a better explanation of how experts and novices are different in their range of eye movement metrics.

These results can be applied to introduce an automatic approach to build a prediction model for expertise with the incorporation of eye movements parameters. We also believe that this result can open the door for new strategies of testing and ranking programmers’ skills. To date, it is a debatable issue of how to decide the expertise level of developers and which way is more accurate such as using years of programming experience

or levels of education. Moreover, studies have shown that expertise is more about cognitive skills and making the right decision that reflect on the experts performance (Shanteau, 1992). Thus, the incorporation of eye-tracking technology can add a more reliable way to assess expertise level depending on how the developer traverses through the visual task. In such a situation, identifying expertise can be a valuable aid to provide appropriate help for students who show less experience in understanding skill than others indicated by, for example, an increase in pupil size, more fixations on some area of the code and longer gaze time, or reading the task line by line.

# Threats to Validity

We note that there is some limitation related to deciding the pupil diameter baseline value for participants. Since the study is conducted by extracting the metrics from a previously published dataset, it is likely that there are other factors that could have possibly affected the pupil diameter size, such as maintaining the light brightness in the experiment setting. Relatedly, to reduce the effect on calculation accuracy and to overcome this limitation, we sample and average participants’ pupil diameter in the first 15 samples of the starting trial to estimate the pupil baseline value rather than using a specific value. We have noticed that the dilation mostly starts after this baseline interval 60 ms (15 samples), as shown in Figure 1 and Figure 2. The risk of having a bias in pupil dilation may arise due to the issue of having different settings. Thus, we calculate the baseline after starting to read the code then studying the changes in the pupil size thorough the task. This step would assure that the cognitive load is the only changing variable between the baseline and finishing the task; the code is presented on a single screen.

As developers read the task for comprehension purposes and as our goal is to assess expertise in a realistic environment, we removed developers with no-programming experience from the analysis. This was decided because they might have strongly affected the time and the number of gaze visits due to the lack of programming experience to put effort into the code.

Participants are considered experts or novices in this study relative to the other participants by taking into account the distribution of the subjects’ programming experience years. Because the mid-point of the developers programming expertise in years is 4 years, it was the chosen threshold to perform the analysis.

In addition, it could be argued that reading the code for comprehension purposes may have affected the way that the participants read the programs. As such, they dedicated more fixations, gaze time, and mental efforts to fully understand the code. However, we could eliminate these factor effects because the subjects did not know before starting the experiment what type of comprehension question they would be given.

# Conclusion

In this chapter, we investigate the use of eye movement-based metrics collected during comprehension tasks to find a distinctive pattern that assesses developers expertise levels (expert/novice). Our findings can be compared to the results of earlier studies that use eye-tracking measures to characterize expertise. We add to that a strong evidence that cognitive load analysis could be a good measurement for expertise level. On average, expertise level has a significant impact on the developers’ comprehension behaviors. Novice programmers exhibit more fixations and longer duration compared to experts. In

applying the analysis of pupil responses between experts and novices, our results show the degree of dilation from the baseline in both groups. These observations conclude that eye movements data contains valuable insights about programmers’ skills. However, since expertise has a significant impact on eye movement-based metrics, it could be a key point for establishing an automatic expertise classification model using developers’ gaze related attributes. Setting our future direction on examining more eye movements metrics can more clearly capture a pattern to distinguish between expert and novice developers. It can also make use of these distinctions in parameters to build and improve the accuracy of an automatic expertise prediction model.

We plan to conduct a more controlled experiment, in the near future, to measure pupil size with respect to expert and novices. Recently, such a study has been impractical to conduct due to COVID-19 restrictions.



# VISUAL AND COGNITIVE EFFORT OVER STIMULI

This chapter compares the results of the first trial to the second one in each group of experts and novices. There are 180 data points for the novice participants (belong to 90 novices who read two tasks, Vehicle.java and Rectangle.java) and 202 data points for experts (that belong to 101 experts who read the same two tasks as novices, Vehicle.java and Rectangle.java). The goal is to compare the studied eye-tracking measurements when reading the first visual activity to the second reading activity to find the changes in the reading patterns that happened through the experiment’s time followed by the experts and novices. This section provides the results to the next research question:

* + - RQ4: Does the order of the task (order in which the stimulus programs were shown to the participants) have an impact on the eye measurements of experts and novices during the comprehension process?

The answer to this question shows the changes in the visual efforts-related metrics and the cognitive load-related metrics between the two stimuli, considering the trial order. In each group, this study used the metrics described in section 5.1.2 to empirically investigate the inter-trial changes of the eye movement parameters in the context of the individual level of expertise.

# Visual Effort Metrics Result by Task Order

The comparative study’s results between the first and the second trial of the fixation-related metrics and the saccade-related metrics are presented in Table 6.1. This

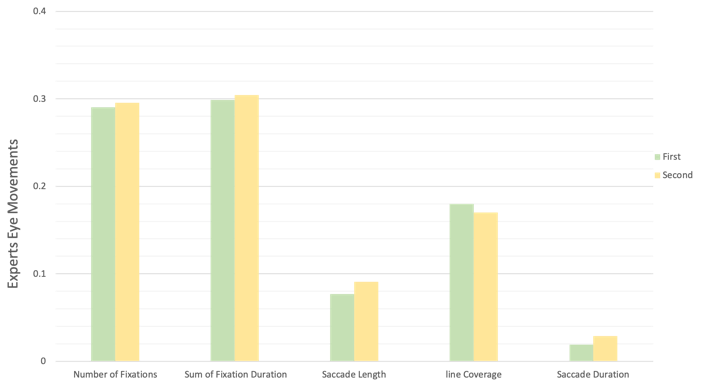
table (Table 6.1) shows the average of the actual data calculated in the two tasks for both the experts and novices.

Then, the significance of these differences in the reading behaviors are tested between the two stimuli using the non-parametric Wilcoxon test (Hollander et al., 2013). Statistical significance is analyzed using the Wilcoxon test, which is deemed appropriate due to the paired data assumption and the data not normally being distributed. Finally, the effect size is calculated using Cohen’s d.

# Table 6.1: The results of the actual data comparing the visual effort of experts and novices in first and second stimuli

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Eye Movement Feature** | **Order of Stimuli** | **Average of Actual Data** | ***SD*** |
| **Experts** | Number of Fixations | First | 12481.07 | 7096.583 |
| Second | 12520.09 | 7950.482 |
| **Novices** | Number of Fixations | First | 14549.77 | 6798.999 |
| Second | 14763.97 | 7322.657 |
| **Experts** | Sum of Fixation Duration | First | 55258.774 ms | 29113.215 |
| Second | 57690.990 ms | 32768.969 |
| **Novices** | Sum of Fixation Duration | First | 65196.439 ms | 30052.643 |
| Second | 65486.234 ms | 30807.716 |
| **Experts** | Saccade Length | First | 4.17 ° | 3.165 |
| Second | 4.89 ° | 5.649 |
| **Novices** | Saccade Length | First | 3.76 ° | 2.697 |
| Second | 4.57 ° | 6.874 |
| **Experts** | Saccade Duration | First | 5.5 ms | 5.457 |
| Second | 6.3 ms | 7.662 |
| **Novices** | Saccade Duration | First | 4.943148 | 4.881 |
| Second | 5.640307 | 8.686 |
| **Experts** | Line Coverage | First | 0.1818682 | 0.108 |
| Second | 0.179618 | 0.093 |
| **Novices** | Line Coverage | First | 0.1842873 | 0.109 |
| Second | 0.1928171 | 0.099 |

* + 1. **Experts’ First vs. Second Stimuli**

Figure 6.1 shows that the experts have a slight increase in their fixation counts and time when reading the second trial versus reading the first one. Similarly, it can be observed that the averages of the saccade length and duration metrics increased over time toward the end of the experiment. However, the experts read fewer code lines in the second visual stimuli than in the first code. The following sections will discuss these results for the expert participants in detail.

(0 ~ 0.4)

# Figure 6.1: Experts’ eye movements comparing the first and second trial

* + - 1. **Fixation Related Metrics Result**

We examine the fixations distributions of the expert participants over the first visual stimuli and then compare them to the devoted fixations later when the second stimuli was shown to them. The actual reading presented in Table 6.1 shows that experts use on average

12481.07 on the first trial, then increase their code visit on the second trial to 12520.09. Same with their gaze time, experts devote more visual attention to the second trial than to the first one. The results show an increase in the gaze time for experts when reading the following visual stimuli by about 2500 ms.

However, based on the computed probability and Z value presented in Table 6.2, these differences in the fixation-related metrics between the two visual tasks are not statistically significant.

# Table 6.2: Wilcoxon test results of experts in fixation related metrics and saccade related metrics comparing the first and second stimuli

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Eye Movement**  **Feature** | ***W*** | ***p*** | ***Z*** | ***d*** |
| **Number of Fixations** | 2725 | 0.6137 | -0.505 | 0.029 |
| **Sum of**  **Fixation Duration** | 2479 | 0.745 | -0.325 | 0.035 |
| **Saccade Length** | 2016 | 0.0804 | -1.748 | 0.16 |
| **Saccade Duration** | 2273 | 0.3063 | -1.023 | 0.12 |
| **Line Coverage** | 2452.5 | 0.678 | -0.415 | 0.022 |

* + - 1. **Saccade Related Metrics Result**

Related to the saccade length metric, the experts show an improvement in their jumping skills through the code lines of the second program. The results obtain from analyzing the second task data (see Table 6.1) present a noticeable increase in the saccade length measurement between the two visual tasks from 4.17° (at the first visual stimuli) to 4.89° (at the second one), respectively. Also, the experts’ average time spent on each saccade increased; the average saccade duration in the second stimuli moved further from

5.5 ms to 6.3 ms.

However, these observations about the increase in the saccade-related metrics are not significant based on the Wilcoxon statistic results presented in Table 6.2.

# Line Coverage Result:

Comparing the percentage of the code line looked at by the expert developers in the two programs in the context of the trial order, Figure 6.1 reveals a slight drop in the number of lines looked at when reading the second trial. Nevertheless, this result is not statistically significant, as shown in Table 6.2.

# Novices’ First vs. Second Stimuli

As shown in Figure 6.2, the novice group also reported a slight increase in all metrics during the second visual stimuli. While experts show less line coverage when reading the second trial, an increase is detected in this metric for the novice participants. Thus, the novices indicate a rise in their visual efforts across all metrics when considering the task order. The following sections provide an explanation for the analysis results.

(0 ~ 0.4)

# Figure 6.2: Novices’ eye movements comparing the first and second trial

* + - 1. **Fixation Related Metrics Result**

The fixation distribution and the gaze time of novices are studied during the first visual trial and then compared to the novices’ responses when reading the second trial. We observe the same result that has been noticed in the expert group. From Table 6.1, the average of the actual reading presents that the novice subjects put more effort and attention toward the end of the experiment. Specifically, the novices have 200 more fixations in the second visual trial, and their gaze time also increases by about 300 ms. However, none of these differences in the fixation-related metrics between the two tasks for the novices are statistically significant, as shown in Table 6.3.

# Table 6.3: Wilcoxon test results of novices in fixation related metrics and saccade related metrics comparing the first and second stimuli. Asterisk indicates significant with 95% confidence. The effect size is given by Cohen’s d.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Eye Movement**  **Feature** | ***W*** | ***p*** | ***Z*** | ***d*** |
| **Number of Fixations** | 1963 | 0.735 | -0.338 | 0.019 |
| **Sum of Fixation**  **Duration** | 1927 | 0.629 | -0.483 | 0.05 |
| **Saccade Length** | 1878 | 0.496 | -0.68 | 0.155 |
| **Saccade Duration** | 1367 | 0.0062\* | -2.736 | 0.129 |
| **Line Coverage** | 1811 | 0.434 | -0.782 | 0.082 |

* + - 1. **Saccade Related Metrics Result**

Like expert subjects, the result shows that the novices, while moving to the second trial, become more familiarized with the task and know where to look over the code areas. Thus, the saccade length measure expands from 3.8 ° to 4.6 ° in the second visual stimuli, as presented in Table 6.1. Along with this increase in the length, the average attention time on each saccade also recorded a significant rise for the novices during reading and comprehending the second code versus the first one (4.9 ms compared to 5.6 ms, respectively).

This difference between the two reading behaviors in the two trials related to the saccade duration is statistically significant (*P* = 0.006, *Z* = -2.736) with a small effect size *d* ≈ 0.13 (Table 6.3).

# Line Coverage Result

Unlike that of the expert participants, the average line viewed by the novices on the subsequent trial is slightly higher than the average line visited in their first visual trial (18% and 19%, respectively), as presented in Table 6.1. However, there is no statistically significant difference between these two reading averages, as shown in Table 6.3.

# Visual Efforts Discussion

Comparing the results of the experts and novices in their eye movement metrics during the comprehension process of the two tasks can reveal insights about the ongoing cognitive process while solving programming problems. For both the experts and novices, results show a slight increase in the visual efforts to understand the second visual task, as revealed in fixation-related measurements. However, these differences between the reading behaviors in the first and second visual stimulus become more noticeable in the saccade- related metrics, as shown in Figure 6.1 and Figure 6.2. Also, in both groups (the experts and novices), saccade length increased while reading and understanding the second trial. This change can explain that individuals are becoming more familiar with the code when moving to the following visual stimuli, making a longer skips and jumping through the code lines knowing where to look in the code areas. However, we observe that these longer saccade lengths are associated with a significantly higher duration for novice developers.

Thus, when novices make long skips within the code, they spend enough time at each saccade to acquire the information they need to comprehend the task. Based on these results, we believe that replication with different types of programming tasks (variety in LOC and difficulty) and performing a timeline analysis of eye movements would better explain the provided high-level observation in this study.

# Cognitive Efforts Metrics Result by Task Order

This section presents the result of the participants’ pupillary responses to the reading and understanding process in the context of the visual stimuli order. Related to the baseline value, we apply the same calculation mentioned in section 5.1.2.2.2 by considering a baseline interval of 60 ms (15 fixations) from the starting trial.

Table 6.4 presents the developers’ cognitive load analysis results using pupil dilation evoked by two visual stimuli tested at four maximum pupil diameter points (0.1, 0.2, 0.3, and 0.4) mm. First, we compare the distribution of the increase occurring in the pupil size above the baseline; for all tested peak values in the two visual stimuli for the two groups (experts and novices). Then, the Wilcoxon non-parametric test is used again to examine the significance of these pupillary response differences. Next, we explain the analysis results of the changes in the cognitive load for experts and novices over the experiment time.

# Table 6.4: The results of the actual data comparing the cognitive effort of experts and novices in first and second stimuli

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Eye**  **Movement Feature** | **Order of stimuli** | **Average of**  **fixation dilation** | **SD** |
| **Experts** | Pupil Size Increase 0.1 | First | 0.15897 | 0.208 |
| Second | 0.11224 | 0.171 |
| **Novices** | Pupil Size Increase 0.1 | First | 0.16972 | 0.211 |
| Second | 0.17207 | 0.241 |
| **Experts** | Pupil Size Increase 0.2 | First | 0.07142 | 0.129 |
| Second | 0.04689 | 0.11 |
| **Novices** | Pupil Size Increase 0.2 | First | 0.08302 | 0.139 |
| Second | 0.08438 | 0.163 |
| **Experts** | Pupil Size Increase 0.3 | First | 0.03134 | 0.089 |
| Second | 0.02467 | 0.076 |
| **Novices** | Pupil Size Increase 0.3 | First | 0.03733 | 0.077 |
| Second | 0.0336 | 0.079 |
| **Experts** | Pupil Size Increase 0.4 | First | 0.01614 | 0.075 |
| Second | 0.01206 | 0.044 |
| **Novices** | Pupil Size Increase 0.4 | First | 0.01524 | 0.039 |
| Second | 0.0117 | 0.029 |

* + 1. **Experts First vs. Second Stimuli**

From the data presented in Figure 6.3, experts reported significantly more pupil dilation evoked by the first visual activity than in the second activity. The results obtained from the empirical analysis of the experts’ pupillary response (see Table 6.4) report a clear decline in the pupil diameter while reading second stimuli. The analysis findings show that the experts have a significantly lower average of fixations with dilated pupils up to 0.1 mm and 0.2 mm. In detail, for the pupil size increase by 0.1 mm over the relative baseline, the results show that the dilated fixations up to this point dropped from 16% at the first visual trial to 11% at the second one. Similarly, in the case of the pupil diameter changing up to

0.2 mm, the analysis shows that 7.1% of the experts’ fixations with pupil size increase up to 0.2 mm while reading the first program. However, this percentage of dilated fixation at the same peak value (0.2 mm) decreased to 4.7% toward the end of the experiment (when reading the second program).

(0 ~ 0.18)

# Figure 6.3: Experts’ pupil dilation comparing the first and second trial

Testing these differences in the cognitive efforts between the two stimuli in the expert group shows that the difference is statistically significant at 0.1 mm and 0.2 mm, as summarized in Table 6.5. From Table 6.5, based on the calculated p and z values, experts experience a statistically significant gradual decline in their cognitive workload over visualizing the comprehension tasks. In terms of 0.1 mm, the average pupil dilation correlated to the second task is significantly lower than the first task (*W* = 3143, *p* = 0.011, *Z* = -2.54), with a medium effect size d ≈ 0.3. As with studying the changes in pupil diameter up to 0.2 mm, the results show a statistically significant difference between the two stimuli (*W*= 2302, *p* = 0.021, *Z*= -2.3034), with a small effect size d = 0.2.

We notice that the differences between the dilation in the pupillary responses to the two visual tasks become less at 0.3 mm and 0.4 mm, as seen in the comparison presented in Figure 6.3. In the case of the pupil diameter which dilated up to 0.3, the experts have few fixations with a dilation up to this value when reading the first code (3.1 %), followed by pupils which contracted over reading the second code that dropped the percentage of the dilated fixations to 2.5 %. On the other hand, the changes in the experts’ pupils increasing to 0.4 mm exhibited in 1.6 % of the total fixations during looking at the first code, followed by a lower average of dilated fixations at the second task of 1.2 %.

However, none of these differences between the changes in pupil diameter (at 0.3 mm and 0.4 mm) in the two visual tasks are statistically significant for the experts, as presented in Table 6.5.

# Table 6.5: Wilcoxon test results of experts in pupil related metrics comparing the first and second stimuli. Asterisks indicate significant with 95% confidence. The effect size is given by Cohen’s d.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pupil Dilation** | ***W*** | ***p*** | ***Z*** | ***d*** |
| **Pupil Size Increase 0.1** | 3143 | 0.01106\* | -2.541 | 0.286 |
| **Pupil Size Increase 0.2** | 2302 | 0.02125\* | -2.303 | 0.205 |
| **Pupil Size Increase 0.3** | 1184 | 0.3372 | -0.959 | 0.08 |
| **Pupil Size Increase 0.4** | 392 | 0.4607 | -0.738 | 0.066 |

* + 1. **Novices First vs. Second Stimuli**

Unlike the experts, the analysis of the pupillary responses to the two visual activities in the novice participants concerning the trial order shows a small difference at all tested peak values, as shown in Figure 6.4. However, we observed a slight increase in the mental efforts while reading the second visual trial, as reflected in the pupil dilation increasing to 0.1 mm and 0.2 mm, as seen in Figure 6.4. From the result reported and Table 6.4, we find that the novice subjects have an average of 16.97% of their total gaze distribution with a pupil dilation up to 0.1 mm while looking at the first stimuli. Then while reading the second stimuli, they exhibited more dilated fixations, reaching 17.2 % of their total code visits. The same slight increase is also noticed at 0.2 mm, in the first trial; on average, 8.3 % out of all the novices’ fixations have a pupil diameter dilated up to 0.2 mm, followed by 8.4 % of the dilated fixations in the second trial.

(0 ~ 0.21)

# Figure 6.4: Novices’ pupil dilation comparing the first and second trial

In addition, testing the dilation in the novices’ pupils caused by the two visual stimuli at 0.3 and 0.4 mm shows more considerable differences in the two pupillary responses (Figure 6.4). In reading the first code, the novices have 3.7 % of their gaze with a pupil size increase of up to 0.3 mm above the baseline while they exhibit a lower average of the dilated fixations up to the same point when reading the second code (3.36%) (Table 6.4). In terms of testing the dilation up to 0.4 mm, the novices record 1.5% of their total fixations, with a dilation response occurring when reading the first visual task. Then, on average, they have 1.2% dilated fixations at the second visual task.

However, none of these differences in the novice group are statistically significant, as shown in Table 6.6.

# Table 6.6: Wilcoxon test results of novices in pupil related metrics comparing the first and second stimuli

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pupil Dilation** | ***W*** | ***p*** | ***Z*** | ***d*** |
| Pupil Size Increase 0.1 | 2182 | 0.2576 | -1.132 | 0.0103 |
| Pupil Size Increase 0.2 | 1619 | 0.5525 | -0.594 | 0.0089 |
| Pupil Size Increase 0.3 | 1040 | 0.4996 | -0.675 | 0.048 |
| Pupil Size Increase 0.4 | 626 | 0.5152 | -0.65 | 0.103 |

* + 1. **Cognitive Efforts Discussion**

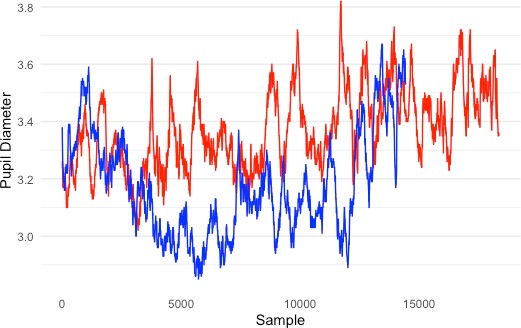
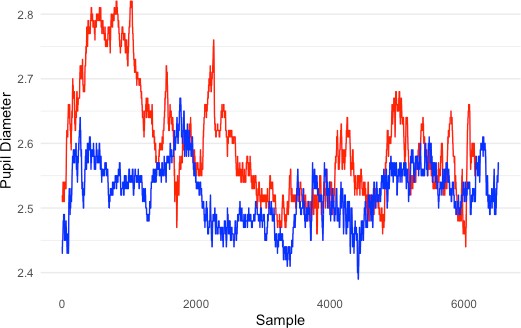
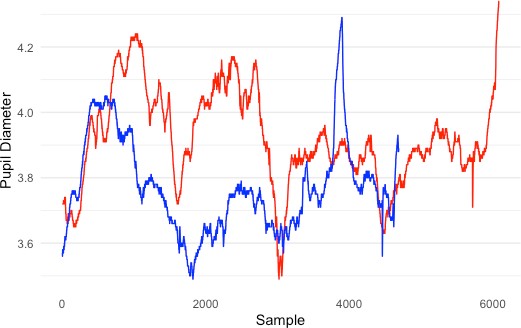
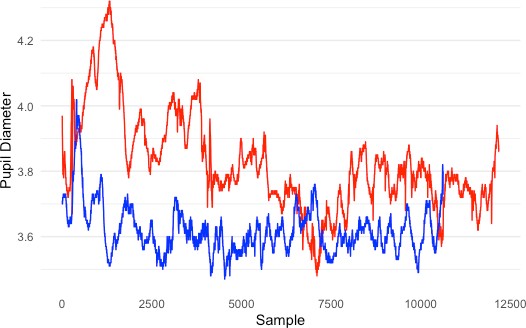
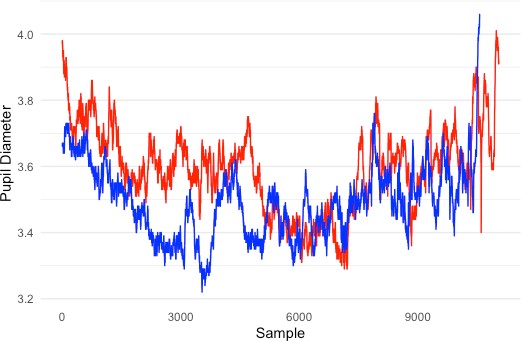
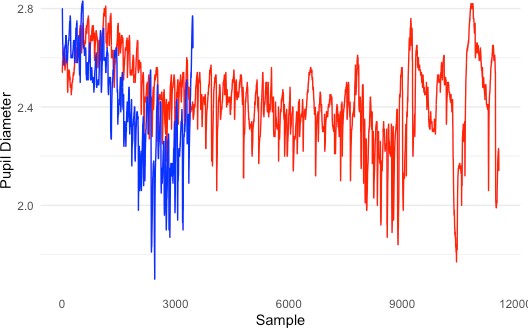
In this analysis of the pupil responses to the two-visual stimulus, we find evidence that cognitive effort corresponds with expertise level and that it tends to change within the experiment, depending on the order of the visual trial. We interpret these changes as follows: first, the experts show a significant drop in the pupil dilation measurements when performing the second comprehension task. Thus, while they become more familiar with the code, as shown in the saccade length metric result, they also become less stressed. Therefore, they put less mental effort into comprehending the second task. This finding confirms those of the previous studies that observed a positive correlation between stress and cognitive effort evaluated by pupil dilation (S. C. Müller & Fritz, 2015; Brown et al., 2018; Behroozi et al., 2018). Second, unlike the experts, we noticed that the novice developers devoted a consistent mental effort to comprehending the two visual activities.

In Figure 6.5 and Figure 6.6, we present an example of the pupil’s timeline for six experts and six novices (respectively) in responses to reading the stimulus (Vehicle.java and Rectangle.java). This timeline shows the pupil diameter changes relative to the baseline value by either dilations or contractions over the two trials. Experts exhibit less cognitive efforts while scanning the second trial, which is shown by a contraction in their pupil diameters as shown in Figure 6.5. However, novices almost have similar cognitive efforts in response to the two trials as seen in Figure 6.6. The red line represents the performing in the first visual trial and the blue line represents the second trial.

# Threats of Validity

Adding to the threats of validity that are discussed in section 5.4 related to calculating the eye movement metrics, this chapter addresses the concern related to the task in this analysis. It could be argued that the task difficulty level or the length may have affected the effort devoted to the visual stimuli in this experiment (not the task order). However, we could eliminate the effects of these factors’ because the two stimulus programs alternate between subjects; they were not present in the same order to all subjects. In addition, the length of the two tasks is similar (Rectangle.java and Vehicle.java, with 18

- 22 LOC). Also, the use of complexity metrics on the code (such as Cyclomatic Complexity) shows that the two programs are simple to understand. Finally, to validate the comparison result, the analysis is performed using the Wilcoxon signed-rank test for paired samples because the data do not follow a normal distribution.



(mm)

(mm)

**( a ) ( b )**

(mm)

(mm)

**( c ) ( d )**

(mm)

(mm)

**( e ) ( f )**

# Figure 6.5: Comparing experts’ pupil dilation timeline when reading the first trial (red) and the second trial (blue), for (a) expert ID-1 and the baseline average is 2.57,

**(b) expert ID-11 and the baseline average is 3.97, (c) expert ID-24 and the baseline average is 3.87, (d) expert ID-190 and the baseline average is 3.72, (e) expert ID-51 and the baseline average is 2.52, (f) expert ID-113 and the baseline average is 3.248**

(mm)

(mm)

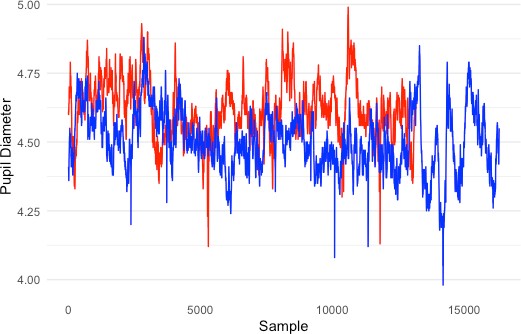
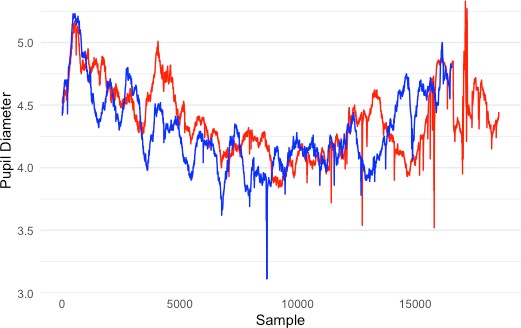
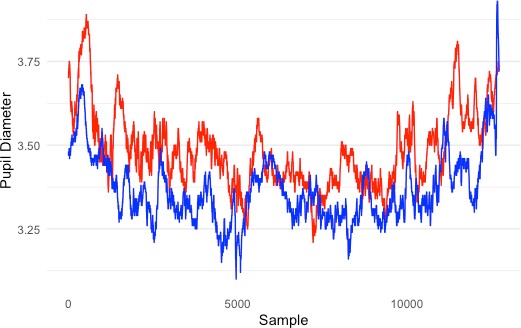
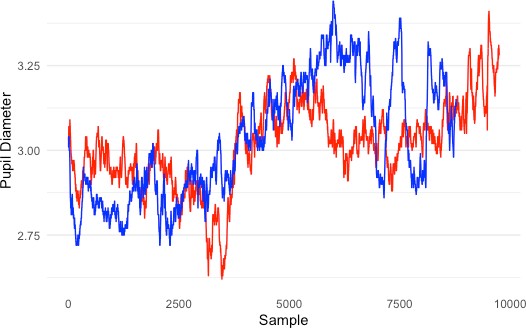
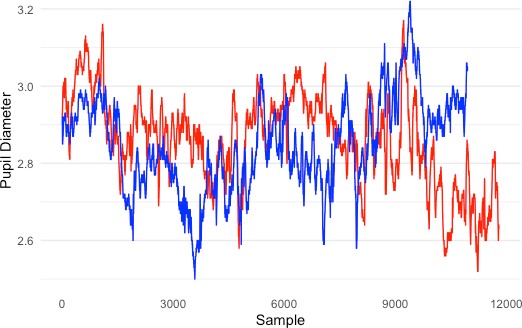
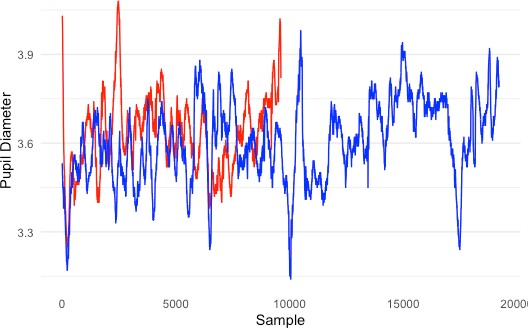
(mm)

(mm)

(mm)

(mm)

# Figure 6.6: Comparing novices’ pupil dilation timeline when reading the first trial (red) and the second trial (blue), for (a) novice ID-6 and the baseline average is 4.012, (b) novice ID-29 and the baseline average is 2.94, (c) novice ID-30 and the baseline average is 3.04, (d) novice ID-31 and the baseline average is 3.718, (e) novice ID-41 and the baseline average is 4.53, (f) novice ID-133 and the baseline average is 4.611



( f )

( e )

( d )

( c )

( b )

( a )

* 1. **Conclusion**

Based on the analyzed eye movements from 207 developers, we conclude that expertise influences reading behaviors during task changes. The study includes the analysis of the devoted visual and mental efforts of experts and novices using eye-related metrics. The result reveals that novices put a consistent reading strategy and attention level (measured by fixations and gaze time) across the two tasks. However, while moving toward the second stimulus, experts show a significantly longer scan strategy while reading over the second code lines. In addition, novices’ pupils exhibit small changes in response to reading the second trial, thus reflecting minor differences in the cognitive load. The evaluation of expertise effects also reveals that experts have significantly less mental effort in response to changing to the following stimulus; they become less stressed as time passes.



# CONCLUSIONS AND FUTURE WORK

The major objective of this dissertation is to improve ways of assessing expertise using a realistic environment by (for example) explaining differences between experts and novices while working in a real setting. This results in a better understanding of developer reading behaviors and how they solve programming tasks. The emerging role of expertise is addressed in two evaluation studies relying on investigating the code reading patterns and evaluating developers’ visual and cognitive efforts during programming. For this purpose, we analyzed two different datasets that have provided two studies that were examined separately (results are not compared or integrated).

The goal of the first study is to provide an in-depth analysis to study programmers’ reading strategies and to explain the differences between experts and novices at a finer level of granularity when reading terms, lines, and statements of the source code. We find that applying measures and analysis from source code is a promising approach to finding distinct differences between experts and novices. Novices show significantly more visits (fixations) to most investigated code areas. The differences between experts and novices in the amount of eye movements over code constructs appear in more fixations when reading keywords, identifiers, method signatures, and variable declarations. Novices also tend to navigate through extra details of the code elements, such as reading names and operators in else statements more heavily than experts during the reviewing process. These results imply that experts show better and more efficient reading skills than novices. They are better at finishing the task using fewer parts of the code while scanning the visual

stimuli. One implication of this result is to be applied to increase a programmer’s productivity by identifying lines and statements that have repeated fixations. That might indicate the difficulty of comprehending these lines, thus providing appropriate help in facilitating code comprehension. In addition, a closer examination of the fixation distributions shows that the programmers visited signatures more frequently than the body. This result confirms the importance of using good naming in method signatures that display the task of the code.

The second evaluation study provides empirical findings in the context of determining the differences between experts’ and novices’ eye movements during the program comprehension process. Several eye-tracking metrics are used to assess the extent to which these measures can distinguish developers with two expertise levels (experts/novices). We use three types of measures to examine the underlying visual and cognitive efforts that developers perform while comprehending the visual stimuli. Together, these measurements outline the critical role of expertise in the devoted attention, focus and mental workload toward performing a programming task. For visual attention differences, studying novices’ eye movements in contrast to experts shows that novices use high amounts of fixations, have longer durations, and cover more lines of the code as a crucial part of facilitating the comprehension process. A possible explanation for this is that novices spend more time to understanding and analyzing the code due to code complexity. This shows the importance of giving beginners enough time for comprehension when teaching programming. However, experts' knowledge of knowing where to look demonstrates their ability of being able to move longer distances between

the code parts, as shown in the significantly longer saccade metric result. This finding further supports the connection between the source code reading strategy and expertise level, which shows the importance of teaching the right way of reading code for beginner programmers (Busjahn et al., 2015).

In addition, cognitive effort evaluation shows a significant impact of expertise reflected in more dilations of novices’ pupils relative to their baseline diameter. We can conclude that pupillary responses to the comprehension process show strong evidence of the cognitive load analysis. We found that all tested pupil diameter values identify significant differences between experts and novices. These findings suggest that novices might see this task as difficult, which was demonstrated by their pupil size increase. Earlier studies have proven that changes in pupil size is the primary metric for deciding the task difficulty (Hess & Polt, 1964; Fritz et al., 2014).

With a further empirical examination of the differences between experts and novices in the devoted cognitive and visual efforts during the comprehension task, we observe the effects of expertise on reading behaviors during the task changes. Both novices and experts show a consistent reading strategy and attention level over the two trials; this is revealed by applying a close average of fixations and total gaze time of both tasks. However, novices show a less linearity scan strategy while reading the second code lines connecting with the cost of significantly more extended time at each saccade to acquire the needed information to comprehend the task. Furthermore, we also observe a significant effect of expertise level on the mental effort while moving to the following trial. Experts’ pupil diameters exhibit a significant contraction while reading and understanding the

second code, thus leading to a significantly lower cognitive load. It seems possible that the result of dedicating less mental effort to the comprehension of the second task was possible due to the experts becoming more familiar with the code, resulting in less stress. This finding confirms previous studies' observations about the positive correlation between stress and cognitive effort, which was evaluated by the pupil dilation (S. C. Müller & Fritz, 2015; Brown et al., 2018; Behroozi et al., 2018).

In future investigations, these are some considerable directions to extend the work. For example, the assessment of developers’ reading behaviors can be improved by considering the gaze time along with fixation counts. Furthermore, instead of a predefined AOI, the approach can include visualizing fixations over the source code using a heatmap to study the special fixation distributions and to find the preferred area on the code.

Additionally, the results of the eye movements metrics will serve as a base for future studies to establish an automatic expertise classification model using developers’ gaze-related attributes. However, more controlled trials could provide additional evidence for assessing the developers’ cognitive workload by combining eye movements with other technologies that measure brain activity, such as using an EEG.

In further research, we also plan on conducting more experiments that focus more on expertise impact on visual and cognitive efforts during task changes. This includes analyzing eye movements data obtained from multiple visual tasks and considering difficulty factors to be defined precisely, as it has been proved to influence other eye parameters like gaze time.

APPENDIX A

# Result of Source Code Terms Coverage for Experts and Novices

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Subject ID** | **Expertise** | **File** | **Method Signatures** | **Keywords** | **Identifiers** | **Operators** |
| **ID1** | Expert | BibtexParser.java | 19.05% | 7.14% | 7.61% | 5.15% |
| **ID2** | Expert | BibtexParser.java | 14.29% | 13.78% | 12.88% | 6.92% |
| **ID3** | Expert | BibtexParser.java | 4.76% | 0.00% | 0.11% | 0.00% |
| **ID4** | Expert | BibtexParser.java | 14.29% | 6.12% | 4.70% | 2.42% |
| **ID5** | Expert | BibtexParser.java | 9.52% | 7.14% | 6.05% | 4.03% |
| **ID6** | Expert | BibtexParser.java | 14.29% | 18.88% | 12.77% | 10.14% |
| **ID7** | Expert | BibtexParser.java | 95.24% | 39.80% | 32.25% | 14.17% |
| **ID8** | Expert | BibtexParser.java | 9.52% | 6.12% | 3.81% | 1.77% |
| **ID9** | Expert | BibtexParser.java | 19.05% | 13.78% | 9.29% | 4.67% |
| **ID11** | Expert | BibtexParser.java | 9.52% | 7.14% | 6.16% | 4.99% |
| **ID22** | Novice | BibtexParser.java | 28.57% | 10.20% | 6.83% | 1.77% |
| **ID23** | Novice | BibtexParser.java | 9.52% | 9.18% | 6.27% | 5.96% |
| **ID24** | Novice | BibtexParser.java | 33.33% | 14.80% | 10.19% | 2.90% |
| **ID25** | Novice | BibtexParser.java | 85.71% | 41.33% | 33.48% | 20.29% |
| **ID26** | Novice | BibtexParser.java | 90.48% | 40.31% | 42.11% | 23.35% |
| **ID27** | Novice | BibtexParser.java | 100.00% | 43.37% | 46.14% | 24.32% |
| **ID30** | Novice | BibtexParser.java | 28.57% | 26.02% | 20.94% | 13.37% |
| **ID31** | Novice | BibtexParser.java | 61.90% | 24.49% | 23.74% | 11.43% |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Subject ID** | **Variable Types** | **Variable Names** | **Operators In-If Statements** | **Names**  **In-If Statements** | **Arguments** |
| **ID1** | 8.43% | 10.47% | 3.96% | 5.63% | 3.13% |
| **ID2** | 18.07% | 15.12% | 5.45% | 8.45% | 9.90% |
| **ID3** | 0.00% | 0.00% | 0.00% | 0.00% | 0.00% |
| **ID4** | 10.84% | 5.81% | 0.99% | 1.41% | 3.13% |
| **ID5** | 7.23% | 8.14% | 1.98% | 7.04% | 2.08% |
| **ID6** | 15.66% | 11.63% | 9.41% | 12.68% | 5.73% |
| **ID7** | 54.22% | 47.67% | 16.34% | 25.35% | 16.15% |
| **ID8** | 6.02% | 6.98% | 0.00% | 1.41% | 1.56% |
| **ID9** | 16.87% | 9.30% | 3.96% | 5.63% | 3.65% |
| **ID11** | 6.02% | 4.65% | 6.93% | 9.86% | 4.17% |
| **ID22** | 15.66% | 9.30% | 0.99% | 1.41% | 1.04% |
| **ID23** | 6.02% | 8.14% | 1.98% | 7.04% | 3.13% |
| **ID24** | 20.48% | 16.28% | 2.48% | 2.82% | 2.08% |
| **ID25** | 36.14% | 37.21% | 18.81% | 23.94% | 18.75% |
| **ID26** | 57.83% | 48.84% | 23.76% | 25.35% | 25.00% |
| **ID27** | 59.04% | 50.00% | 15.84% | 21.13% | 29.69% |
| **ID30** | 32.53% | 29.07% | 15.35% | 14.08% | 8.33% |
| **ID31** | 31.33% | 24.42% | 6.44% | 11.27% | 13.54% |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Subject ID** | **Names in-While**  **Statements** | **Operators in-While**  **Statements** | **Names in-Else**  **Statements** | **Operators in-Else**  **Statements** |
| **ID1** | 14.29% | 9.21% | 14.29% | 14.29% |
| **ID2** | 7.14% | 10.53% | 14.29% | 0.00% |
| **ID3** | 0.00% | 0.00% | 0.00% | 0.00% |
| **ID4** | 14.29% | 3.95% | 0.00% | 0.00% |
| **ID5** | 7.14% | 1.32% | 7.14% | 0.00% |
| **ID6** | 14.29% | 25.00% | 14.29% | 0.00% |
| **ID7** | 21.43% | 13.16% | 14.29% | 0.00% |
| **ID8** | 14.29% | 6.58% | 0.00% | 0.00% |
| **ID9** | 7.14% | 6.58% | 0.00% | 0.00% |
| **ID11** | 7.14% | 7.89% | 0.00% | 0.00% |
| **ID22** | 7.14% | 1.32% | 7.14% | 0.00% |
| **ID23** | 21.43% | 26.32% | 14.29% | 14.29% |
| **ID24** | 0.00% | 0.00% | 14.29% | 14.29% |
| **ID25** | 28.57% | 27.63% | 35.71% | 28.57% |
| **ID26** | 21.43% | 11.84% | 21.43% | 14.29% |
| **ID27** | 42.86% | 32.89% | 21.43% | 0.00% |
| **ID30** | 21.43% | 10.53% | 14.29% | 14.29% |
| **ID31** | 21.43% | 19.74% | 7.14% | 0.00% |

**REFERENCES**

Abid, N. J., Maletic, J. I., & Sharif, B. (2019). Using Developer Eye Movements to Externalize the Mental Model Used in Code Summarization Tasks. *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, 13:1- 13:9. https://doi.org/10.1145/3314111.3319834

Abid, N. J., Sharif, B., Dragan, N., Alrasheed, H., & Maletic, J. I. (2019). Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 384–395. https://doi.org/10.1109/ICSE.2019.00052

Ahrens, M., Schneider, K., & Busch, M. (2019). Attention in Software Maintenance: An Eye Tracking Study. *2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)*, 2–9. https://doi.org/10.1109/EMIP.2019.00009

Al Madi, N., Peterson, C., Sharif, B., & Maletic, J. (2021). *From Novice to Expert: Analysis of Token Level Effects in a Longitudinal Eye Tracking Study*. https://doi.org/10.1109/ICPC52881.2021.00025

Ali, N., Sharafi, Z., Guéhéneuc, Y.-G., & Antoniol, G. (2015). An empirical study on the importance of source code entities for requirements traceability. *Empirical Software Engineering*, *20*(2), 442–478. https://doi.org/10.1007/s10664-014-9315- y

Aljehane, S., Sharif, B., & Maletic, J. (2021). Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on

Source Code Constructs During a Bug Fixing Task. In *ACM Symposium on Eye Tracking Research and Applications* (pp. 1–6). Association for Computing Machinery. https://doi.org/10.1145/3448018.3457424

Alqadi, B. S., & Maletic, J. I. (2017). An Empirical Study of Debugging Patterns Among Novices Programmers. *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*, 15–20. https://doi.org/10.1145/3017680.3017761

Arisholm, E., Gallis, H., Dyba, T., & Sjoberg, D. I. K. (2007). Evaluating Pair Programming with Respect to System Complexity and Programmer Expertise. *IEEE Transactions on Software Engineering*, *33*(2), 65–86. https://doi.org/10.1109/TSE.2007.17

Anders Ericsson, K. (2008). Deliberate Practice and Acquisition of Expert Performance: A General Overview. Academic Emergency Medicine, 15(11), 988–994. https://doi.org/10.1111/j.1553-2712.2008.00227.x

Bailey, B. P., & Iqbal, S. T. (2008). Understanding changes in mental workload during execution of goal-directed tasks and its application for interruption management. *ACM Transactions on Computer-Human Interaction*, *14*(4), 1–28. https://doi.org/10.1145/1314683.1314689

Beatty, J. (1982). Task-evoked pupillary responses, processing load, and the structure of processing resources. *Psychological Bulletin*, *91*(2), 276–292. https://doi.org/10.1037/0033-2909.91.2.276

Beatty, J., & Lucero-Wagoner, B. (2000). The pupillary system. In *Handbook of psychophysiology, 2nd ed* (pp. 142–162). Cambridge University Press.

Bednarik, R. (2012). Expertise-dependent visual attention strategies develop over time during debugging with multiple code representations. *International Journal of Human-Computer Studies*, *70*(2), 143–155. https://doi.org/10.1016/j.ijhcs.2011.09.003

Bednarik, R., Busjahn, T., Gibaldi, A., Ahadi, A., Bielikova, M., Crosby, M., Essig, K., Fagerholm, F., Jbara, A., Lister, R., Orlov, P., Paterson, J., Sharif, B., Sirkiä, T., Stelovsky, J., Tvarozek, J., Vrzakova, H., & van der Linde, I. (2020). EMIP: The eye movements in programming dataset. *Science of Computer Programming*, *198*, 102520. https://doi.org/10.1016/j.scico.2020.102520

Bednarik, R., & Tukiainen, M. (2006). An eye-tracking methodology for characterizing program comprehension processes. *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications*, 125–132. https://doi.org/10.1145/1117309.1117356

Begel, A., & Vrzakova, H. (2018). Eye movements in code review. *Proceedings of the Workshop on Eye Movements in Programming - EMIP ’18*, 1–5. https://doi.org/10.1145/3216723.3216727

Behroozi, M., Lui, A., Moore, I., Ford, D., & Parnin, C. (2018). Dazed: Measuring the cognitive load of solving technical interview problems at the whiteboard.

*Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 93–96. https://doi.org/10.1145/3183399.3183415

Biffl, S., & Grossmann, W. (2001). Evaluating the accuracy of defect estimation models based on inspection data from two inspection cycles. *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, 145–154. https://doi.org/10.1109/ICSE.2001.919089

Binkley, D., Davis, M., Lawrie, D., Maletic, J. I., Morrell, C., & Sharif, B. (2013). The impact of identifier style on effort and comprehension. *Empirical Software Engineering*, *18*(2), 219–276. https://doi.org/10.1007/s10664-012-9201-4

Brooks, R. (1983). Towards a theory of the comprehension of computer programs.

*International Journal of Man-Machine Studies*, *18*(6), 543–554. https://doi.org/10.1016/S0020-7373(83)80031-5

Brown, J. A., Ivanov, V., Rogers, A., Succi, G., Tormasov, A., & Yi, J. (2018). *Toward a Better Understanding of How to Develop Software Under Stress—Drafting the Lines for Future Research*. https://doi.org/10.48550/arXiv.1804.09044

Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J. H., Schulte, C., Sharif, B., & Tamm, S. (2015). Eye Movements in Code Reading: Relaxing the Linear Order. *2015 IEEE 23rd International Conference on Program Comprehension*, 255–265. https://doi.org/10.1109/ICPC.2015.36

Busjahn, T., Bednarik, R., & Schulte, C. (2014). What influences dwell time during source code reading? Analysis of element type and frequency as factors.

*Proceedings of the Symposium on Eye Tracking Research and Applications*, 335– 338. https://doi.org/10.1145/2578153.2578211

Busjahn, T., Schulte, C., & Busjahn, A. (2011). Analysis of code reading to gain more insight in program comprehension. *Proceedings of the 11th Koli Calling International Conference on Computing Education Research - Koli Calling ’11*, 1. https://doi.org/10.1145/2094131.2094133

Cagiltay, N. E., Tokdemir, G., Kilic, O., & Topalli, D. (2013). Performing and analyzing non-formal inspections of entity relationship diagram (ERD). *Journal of Systems and Software*, *86*(8), 2184–2195. https://doi.org/10.1016/j.jss.2013.03.106

Carniglia, E., Caputi, M., Manfredi, V., Zambarbieri, D., & Pessa, E. (2012). The influence of emotional picture thematic content on exploratory eye movements. *Journal of Eye Movement Research*, *5*(4), Article 4. https://doi.org/10.16910/jemr.5.4.4

Cepeda Porras, G., & Guéhéneuc, Y.-G. (2010). An empirical study on the efficiency of different design pattern representations in UML class diagrams. *Empirical Software Engineering*, *15*(5), 493–522. https://doi.org/10.1007/s10664-009-9125-

9

Collard, M. L., Decker, M. J., & Maletic, J. I. (2011). Lightweight Transformation and Fact Extraction with the srcML Toolkit. *2011 IEEE 11th International Working Conference on Source Code Analysis and Manipulation*, 173–184. https://doi.org/10.1109/SCAM.2011.19

Couceiro, R., Barbosa, R., Duráes, J., Duarte, G., Castelhano, J., Duarte, C., Teixeira, C., Laranjeiro, N., Medeiros, J., Carvalho, P., Castelo Branco, M., & Madeira, H. (2019). Spotting Problematic Code Lines using Nonintrusive Programmers’

Biofeedback. *2019 IEEE 30th International Symposium on Software Reliability Engineering (ISSRE)*, 93–103. https://doi.org/10.1109/ISSRE.2019.00019

Couceiro, R., Duarte, G., Durães, J., Castelhano, J., Duarte, C., Teixeira, C., Castelo Branco, M., Carvalho, P., & Madeira, H. (2019a). Biofeedback Augmented Software Engineering: Monitoring of Programmers’ Mental Effort. *2019 IEEE/ACM 41st International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*, 37–40. https://doi.org/10.1109/ICSE- NIER.2019.00018

Couceiro, R., Duarte, G., Durães, J., Castelhano, J., Duarte, C., Teixeira, C., Castelo Branco, M., Carvalho, P., & Madeira, H. (2019b). Pupillography as Indicator of Programmers’ Mental Effort and Cognitive Overload. *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 638–644. https://doi.org/10.1109/DSN.2019.00069

Crosby, M. E., Scholtz, J., & Wiedenbeck, S. (2002). *The Roles Beacons Play in Comprehension for Novice and Expert Programmers*. 10.

Crosby, M. E., & Stelovsky, J. (1990). How do we read algorithms? A case study.

*Computer*, *23*(1), 25–35. https://doi.org/10.1109/2.48797

De Smet, B., Lempereur, L., Sharafi, Z., Guéhéneuc, Y.-G., Antoniol, G., & Habra, N. (2014). Taupe: Visualizing and analyzing eye-tracking data. *Science of Computer Programming*, *79*, 260–278. https://doi.org/10.1016/j.scico.2012.01.004

Deitelhoff, F., Harrer, A., & Kienle, A. (2019). The Influence of Different AOI Models in Source Code Comprehension Analysis. *2019 IEEE/ACM 6th International*

*Workshop on Eye Movements in Programming (EMIP)*, 10–17. https://doi.org/10.1109/EMIP.2019.00010

Eckstein, M. K., Guerra-Carrillo, B., Miller Singley, A. T., & Bunge, S. A. (2017).

Beyond eye gaze: What else can eyetracking reveal about cognition and cognitive development? *Developmental Cognitive Neuroscience*, *25*, 69–91. https://doi.org/10.1016/j.dcn.2016.11.001

Ericsson, K., & Lehmann, A. (1996). Expert and Exceptional Performance: Evidence of Maximal Adaptation to Task Constraints. *Annual Review of Psychology*, *47*, 273–

305. https://doi.org/10.1146/annurev.psych.47.1.273

Fakhoury, S., Ma, Y., Arnaoudova, V., & Adesope, O. (2018). The Effect of Poor Source Code Lexicon and Readability on Developers’ Cognitive Load. *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 286–28610.

Feigenspan, J., Kästner, C., Liebig, J., Apel, S., & Hanenberg, S. (2012). Measuring programming experience. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 73–82. https://doi.org/10.1109/ICPC.2012.6240511

Feigenspan, J., Schulze, M., Papendieck, M., Kastner, C., Dachselt, R., Koppen, V., & Frisch, M. (2011). Using background colors to support program comprehension in software product lines. *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011)*, 66–75. https://doi.org/10.1049/ic.2011.0008

Fritz, T., Begel, A., Müller, S. C., Yigit-Elliott, S., & Züger, M. (2014). Using Psycho- physiological Measures to Assess Task Difficulty in Software Development.

*Proceedings of the 36th International Conference on Software Engineering*, 402– 413. https://doi.org/10.1145/2568225.2568266

Goldberg, J. H., & Kotval, X. P. (1999). Computer interface evaluation using eye movements: Methods and constructs. *International Journal of Industrial Ergonomics*, *24*(6), 631–645. https://doi.org/10.1016/S0169-8141(98)00068-7

Goldinger, S. D., & Papesh, M. H. (2012). Pupil Dilation Reflects the Creation and Retrieval of Memories. *Current Directions in Psychological Science*, *21*(2), 90– 95. https://doi.org/10.1177/0963721412436811

Granholm, E., & Steinhauer, S. R. (Eds.). (2004). Pupillometric measures of cognitive and emotional processes. *International Journal of Psychophysiology*, *52*(1), 1–6. https://doi.org/10.1016/j.ijpsycho.2003.12.001

Guarnera, D. T., Bryant, C. A., Mishra, A., Maletic, J. I., & Sharif, B. (2018). iTrace: Eye tracking infrastructure for development environments. *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, 1–3. https://doi.org/10.1145/3204493.3208343

Guéhéneuc, Y.-G. (2006). TAUPE: Towards Understanding Program Comprehension.

*Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research*. https://doi.org/10.1145/1188966.1188968

Hakerem, G., & Sutton, S. (1966). Pupillary Response at Visual Threshold. *Nature*, *212*(5061), 485–486. https://doi.org/10.1038/212485a0

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.*, *11*(1), 10–18. https://doi.org/10.1145/1656274.1656278

Hess, E. H., & Polt, J. M. (1964). Pupil Size in Relation to Mental Activity during Simple Problem-Solving. *Science*, *143*(3611), 1190–1192. https://doi.org/10.1126/science.143.3611.1190

Hoeks, B., & Levelt, W. J. M. (1993). Pupillary dilation as a measure of attention: A quantitative system analysis. *Behavior Research Methods, Instruments, & Computers*, *25*(1), 16–26. https://doi.org/10.3758/BF03204445

Hofmeister, J., Siegmund, J., & Holt, D. V. (2017). Shorter identifier names take longer to comprehend. *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 217–227. https://doi.org/10.1109/SANER.2017.7884623

Hogervorst, M. A., Brouwer, A.-M., & van Erp, J. B. F. (2014). Combining and comparing EEG, peripheral physiology and eye-related measures for the assessment of mental workload. *Frontiers in Neuroscience*, *8*. https://doi.org/10.3389/fnins.2014.00322

Holland, M. K., & Tarlow, G. (1972). Blinking and Mental Load. *Psychological Reports*, *31*(1), 119–127. https://doi.org/10.2466/pr0.1972.31.1.119

Holland, M. K., & Tarlow, G. (1975). Blinking and Thinking. *Perceptual and Motor Skills*, *41*(2), 403–406. https://doi.org/10.2466/pms.1975.41.2.403

Hollander, M., Wolfe, D. A., & Chicken, E. (2013). *Nonparametric Statistical Methods*.

John Wiley & Sons.

Iqbal, S. T., Zheng, X. S., & Bailey, B. P. (2004). Task-evoked pupillary response to mental workload in human-computer interaction. *Extended Abstracts of the 2004 Conference on Human Factors and Computing Systems - CHI ’04*, 1477. https://doi.org/10.1145/985921.986094

İşbilir, E., Çakır, M. P., Acartürk, C., & Tekerek, A. Ş. (2019). Towards a Multimodal Model of Cognitive Workload Through Synchronous Optical Brain Imaging and Eye Tracking Measures. *Frontiers in Human Neuroscience*, *13*. https://[www.frontiersin.org/article/10.3389/fnhum.2019.00375](http://www.frontiersin.org/article/10.3389/fnhum.2019.00375)

Jacob, R. J. K., & Karn, K. S. (2003). Commentary on Section 4 - Eye Tracking in Human-Computer Interaction and Usability Research: Ready to Deliver the Promises. In J. Hyönä, R. Radach, & H. Deubel (Eds.), *The Mind’s Eye* (pp. 573– 605). North-Holland. https://doi.org/10.1016/B978-044451020-4/50031-1

Jimenez-Molina, A., Retamal, C., & Lira, H. (2018). Using Psychophysiological Sensors to Assess Mental Workload During Web Browsing. *Sensors*, *18*(2), 458. https://doi.org/10.3390/s18020458

Kahneman, D., & Beatty, J. (1966). Pupil Diameter and Load on Memory. *Science*, *154*(3756), 1583–1585.

Kevic, K., Walters, B. M., Shaffer, T. R., Sharif, B., Shepherd, D. C., & Fritz, T. (2015).

Tracing Software Developers’ Eyes and Interactions for Change Tasks.

*Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 202–213. https://doi.org/10.1145/2786805.2786864

Kevic, K., Walters, B. M., Shaffer, T. R., Sharif, B., Shepherd, D. C., & Fritz, T. (2017). Eye gaze and interaction contexts for change tasks – Observations and potential. *Journal of Systems and Software*, *128*, 252–266. https://doi.org/10.1016/j.jss.2016.03.030

Kiefer, P., Giannopoulos, I., Duchowski, A., & Raubal, M. (2016). Measuring Cognitive Load for Map Tasks Through Pupil Diameter. In J. A. Miller, D. O’Sullivan, & N. Wiegand (Eds.), *Geographic Information Science* (pp. 323–337). Springer International Publishing. https://doi.org/10.1007/978-3-319-45738-3\_21

Klingner, J. (2010). Fixation-aligned pupillary response averaging. *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications - ETRA ’10*, 275. https://doi.org/10.1145/1743666.1743732

Klingner, J., Tversky, B., & Hanrahan, P. (2011). Effects of visual and verbal presentation on cognitive load in vigilance, memory, and arithmetic tasks. *Psychophysiology*, *48*(3), 323–332. https://doi.org/10.1111/j.1469- 8986.2010.01069.x

Kontogiorgos, D., & Manikas, K. (2015). *Towards identifying programming expertise with the use of physiological measures*. Eye Movements in Programming. [http://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-194351](http://urn.kb.se/resolve?urn=urn%3Anbn%3Ase%3Akth%3Adiva-194351)

Kuric, E., & Bieliková, M. (2014). Estimation of student’s programming expertise.

*Proceedings of the 8th ACM/IEEE International Symposium on Empirical*

*Software Engineering and Measurement*, 1–4. https://doi.org/10.1145/2652524.2652561

Lawrie, D., Morrell, C., Feild, H., & Binkley, D. (2007). Effective identifier names for comprehension and memory. *Innovations in Systems and Software Engineering*, *3*(4), 303–318. https://doi.org/10.1007/s11334-007-0031-2

Lee, S., Matteson, A., Hooshyar, D., Kim, S., Jung, J., Nam, G., & Lim, H. (2016).

Comparing Programming Language Comprehension between Novice and Expert Programmers Using EEG Analysis. *2016 IEEE 16th International Conference on Bioinformatics and Bioengineering (BIBE)*, 350–355. https://doi.org/10.1109/BIBE.2016.30

Madi, N. A., Peterson, C. S., Sharif, B., & Maletic, J. I. (2021). *From Novice to Expert: Analysis of Token Level Effects in a Longitudinal Eye Tracking Study*. 172–183. https://doi.org/10.1109/ICPC52881.2021.00025

Marshall, S. P. (2002). The Index of Cognitive Activity: Measuring cognitive workload. *Proceedings of the IEEE 7th Conference on Human Factors and Power Plants*, 7–7. https://doi.org/10.1109/HFPP.2002.1042860

Müller, M. M. (2004). Are Reviews an Alternative to Pair Programming? *Empirical Software Engineering*, *9*(4), 335–351. https://doi.org/10.1023/B:EMSE.0000039883.47173.39

Müller, S. C., & Fritz, T. (2015). Stuck and Frustrated or in Flow and Happy: Sensing Developers’ Emotions and Progress. *2015 IEEE/ACM 37th IEEE International*

*Conference on Software Engineering*, *1*, 688–699. https://doi.org/10.1109/ICSE.2015.334

Müller, S. C., & Fritz, T. (2016). Using (Bio)Metrics to Predict Code Quality Online.

*2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, 452–463. https://doi.org/10.1145/2884781.2884803

Palinko, O., Kun, A. L., Shyrokov, A., & Heeman, P. (2010). Estimating cognitive load using remote eye tracking in a driving simulator. *Proceedings of the 2010 Symposium on Eye-Tracking Research & Applications - ETRA ’10*, 141. https://doi.org/10.1145/1743666.1743701

Peitek, N., Siegmund, J., Apel, S., Kästner, C., Parnin, C., Bethmann, A., Leich, T., Saake, G., & Brechmann, A. (2020). A Look into Programmers’ Heads. *IEEE Transactions on Software Engineering*, *46*(4), 442–462. https://doi.org/10.1109/TSE.2018.2863303

Peterson, C. S., Abid, N. J., Bryant, C. A., Maletic, J. I., & Sharif, B. (2019). Factors influencing dwell time during source code reading: A large-scale replication experiment. *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, 1–4. https://doi.org/10.1145/3314111.3319833

Petrusel, R., & Mendling, J. (2013). Eye-Tracking the Factors of Process Model Comprehension Tasks. In C. Salinesi, M. C. Norrie, & Ó. Pastor (Eds.), *Advanced Information Systems Engineering* (pp. 224–239). Springer. https://doi.org/10.1007/978-3-642-38709-8\_15

Poole, A., & Ball, L. J. (2005). Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future. *Prospects”, Chapter in C. Ghaoui (Ed.): Encyclopedia of Human-Computer Interaction. Pennsylvania: Idea Group, Inc*.

Poole, A., Ball, L. J., & Phillips, P. (2005). In Search of Salience: A Response-time and Eye-movement Analysis of Bookmark Recognition. In S. Fincher, P. Markopoulos, D. Moore, & R. Ruddle (Eds.), *People and Computers XVIII — Design for Life* (pp. 363–378). Springer London. https://doi.org/10.1007/1-84628- 062-1\_23

Ricca, F., Di Penta, M., Torchiano, M., Tonella, P., & Ceccato, M. (2007). *The Role of Experience and Ability in Comprehension Tasks Supported by UML Stereotypes* (p. 384). https://doi.org/10.1109/ICSE.2007.86

Rodeghero, P., & McMillan, C. (2015). An Empirical Study on the Patterns of Eye Movement during Summarization Tasks. *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, 1–10. https://doi.org/10.1109/ESEM.2015.7321188

Rodeghero, P., McMillan, C., McBurney, P. W., Bosch, N., & D’Mello, S. (2014).

Improving automated source code summarization via an eye-tracking study of programmers. *Proceedings of the 36th International Conference on Software Engineering - ICSE 2014*, 390–401. https://doi.org/10.1145/2568225.2568247

Scalabrino, S., Bavota, G., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D., & Oliveto, R. (2017). Automatically assessing code understandability: How far are

we? *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 417–427. https://doi.org/10.1109/ASE.2017.8115654

Scalabrino, S., Linares-Vásquez, M., Poshyvanyk, D., & Oliveto, R. (2016). Improving code readability models with textual features. *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, 1–10. https://doi.org/10.1109/ICPC.2016.7503707

Schober, P., Boer, C., & Schwarte, L. A. (2018). Correlation Coefficients: Appropriate Use and Interpretation. *Anesthesia & Analgesia*, *126*(5), 1763–1768. https://doi.org/10.1213/ANE.0000000000002864

Shaffer, T. R., Wise, J. L., Walters, B. M., Müller, S. C., Falcone, M., & Sharif, B. (2015). iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 954–957. https://doi.org/10.1145/2786805.2803188

Shanteau, J. (1992). Competence in experts: The role of task characteristics.

*Organizational Behavior and Human Decision Processes*, *53*(2), 252–266. https://doi.org/10.1016/0749-5978(92)90064-E

Sharafi, Z., Marchetto, A., Susi, A., Antoniol, G., & Guéhéneuc, Y. (2013). An empirical study on the efficiency of graphical vs. Textual representations in requirements comprehension. *2013 21st International Conference on Program Comprehension (ICPC)*, 33–42. https://doi.org/10.1109/ICPC.2013.6613831

Sharafi, Z., Shaffer, T., Sharif, B., & Guéhéneuc, Y. (2015). Eye-Tracking Metrics in Software Engineering. *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 96–103. https://doi.org/10.1109/APSEC.2015.53

Sharafi, Z., Sharif, B., Guéhéneuc, Y.-G., Begel, A., Bednarik, R., & Crosby, M. (2020).

A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, *25*(5), 3128–3174. https://doi.org/10.1007/s10664-020-09829-4

Sharafi, Z., Soh, Z., & Guéhéneuc, Y.-G. (2015). A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology*, *67*, 79–107. https://doi.org/10.1016/j.infsof.2015.06.008

Sharif, B. (2011). Empirical assessment of UML class diagram layouts based on architectural importance. *2011 27th IEEE International Conference on Software Maintenance (ICSM)*, 544–549. https://doi.org/10.1109/ICSM.2011.6080828

Sharif, B., Falcone, M., & Maletic, J. I. (2012). An Eye-tracking Study on the Role of Scan Time in Finding Source Code Defects. *Proceedings of the Symposium on Eye Tracking Research and Applications*, 381–384. https://doi.org/10.1145/2168556.2168642

Sharif, B., & Maletic, J. I. (2010a). An Eye Tracking Study on camelCase and under\_score Identifier Styles. *2010 IEEE 18th International Conference on Program Comprehension*, 196–205. https://doi.org/10.1109/ICPC.2010.41

Sharif, B., & Maletic, J. I. (2010b). An eye tracking study on the effects of layout in understanding the role of design patterns. *2010 IEEE International Conference on Software Maintenance*, 1–10. https://doi.org/10.1109/ICSM.2010.5609582

Shrestha, N., Botta, C., Barik, T., & Parnin, C. (2020). Here We Go Again: Why Is It Difficult for Developers to Learn Another Programming Language? *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, 691– 701.

Siegmund, J., Kästner, C., Liebig, J., Apel, S., & Hanenberg, S. (2014). Measuring and modeling programming experience. *Empirical Software Engineering*, *19*(5), 1299–1334. https://doi.org/10.1007/s10664-013-9286-4

Siegmund, J., Peitek, N., Parnin, C., Apel, S., Hofmeister, J., Kästner, C., Begel, A., Bethmann, A., & Brechmann, A. (2017). Measuring neural efficiency of program comprehension. *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 140–150. https://doi.org/10.1145/3106237.3106268

Sillito, J., Murphy, G. C., & De Volder, K. (2008). Asking and Answering Questions during a Programming Change Task. *IEEE Transactions on Software Engineering*, *34*(4), 434–451. https://doi.org/10.1109/TSE.2008.26

Sirois, S., & Brisson, J. (2014). Pupillometry. *WIREs Cognitive Science*, *5*(6), 679–692. https://doi.org/10.1002/wcs.1323

Soh, Z., Sharafi, Z., Van den Plas, B., Porras, G. C., Guéhéneuc, Y.-G., & Antoniol, G. (2012). Professional status and expertise for UML class diagram comprehension:

An empirical study. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 163–172. https://doi.org/10.1109/ICPC.2012.6240484

Soloway, E., & Ehrlich, K. (1984). Empirical Studies of Programming Knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609. https://doi.org/10.1109/TSE.1984.5010283

Telford, C. W., & Thompson, N. (1933). Some factors influencing voluntary and reflex eyelid responses. *Journal of Experimental Psychology*, *16*(4), 524–539. https://doi.org/10.1037/h0071694

Tsukahara, J. S., Harrison, T. L., & Engle, R. W. (2016). The relationship between baseline pupil size and intelligence. *Cognitive Psychology*, *91*, 109–123. https://doi.org/10.1016/j.cogpsych.2016.10.001

Turner, R., Falcone, M., Sharif, B., & Lazar, A. (2014). An Eye-tracking Study Assessing the Comprehension of C++ and Python Source Code. *Proceedings of the Symposium on Eye Tracking Research and Applications*, 231–234. https://doi.org/10.1145/2578153.2578218

Uwano, H., Nakamura, M., Monden, A., & Matsumoto, K. (2006). Analyzing individual performance of source code review using reviewers’ eye movement. *Proceedings of the 2006 Symposium on Eye Tracking Research & Applications - ETRA ’06*, 133. https://doi.org/10.1145/1117309.1117357

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis.

*International Journal of Man-Machine Studies*, *23*(5), 459–494. https://doi.org/10.1016/S0020-7373(85)80054-7

Wulff-Jensen, A., Ruder, K., Triantafyllou, E., & Bruni, L. E. (2019). Gaze Strategies Can Reveal the Impact of Source Code Features on the Cognitive Load of Novice Programmers. In H. Ayaz & L. Mazur (Eds.), *Advances in Neuroergonomics and Cognitive Engineering* (pp. 91–100). Springer International Publishing. https://doi.org/10.1007/978-3-319-94866-9\_9

Yu, C. S., Treude, C., & Aniche, M. (2019). Comprehending Test Code: An Empirical Study. *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 501–512. https://doi.org/10.1109/ICSME.2019.00084

Yusuf, S., Kagdi, H., & Maletic, J. I. (2007). Assessing the Comprehension of UML Class Diagrams via Eye Tracking. *15th IEEE International Conference on Program Comprehension (ICPC ’07)*, 113–122. https://doi.org/10.1109/ICPC.2007.10

Züger, M., & Fritz, T. (2015). Interruptibility of Software Developers and its Prediction Using Psycho-Physiological Sensors. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2981–2990. https://doi.org/10.1145/2702123.2702593