Determining Differences in Reading Behavior Between Experts and Novices by Investigating Eye Movement on Source Code Constructs During a Bug Fixing Task

Salwa, D., Aljehane

Kent State University, Department of Computer Science, saljehan@kent.edu

Bonita, Sharif

University of Nebraska - Lincoln, Department of Computer Science and Engineering, bsharif@unl.edu

Jonathan, I., Maletic

Kent State University, Department of Computer Science, jmaletic@kent.edu

This research compares the eye movement of expert and novice programmers working on a bug fixing task. This comparison aims at investigating which source code elements programmers focus on when they review Java source code. Programmer code reading behaviors at the line and term levels are used to characterize the differences between experts and novices. The study analyzes programmers eye movements over identified source code areas using an existing eye tracking dataset of 12 experts and 10 novices. The results show that the difference between experts and novices is significant in source code element coverage. Specifically, novices read more method signatures, variable declarations, identifiers, and keywords compared to experts. However, experts are better at finishing the task using fewer source code elements when compared to novices. Moreover, programmers tend to focus on the method signatures the most while reading the code.

CCS CONCEPTS • **Human-centered computing → Human computer interaction (HCI) → Empirical studies in HCI • Software and its engineering** → *General programming languages*

Additional Keywords and Phrases: eye tracking study, eye movement analysis, source code reading, token reading, expertise

ACM Reference Format:

First Author's Name, Initials, and Last Name, Second Author's Name, Initials, and Last Name, and Third Author's Name, Initials, and Last Name. 2018. The Title of the Paper: ACM Conference Proceedings Manuscript Submission Template: This is the subtitle of the paper, this document both explains and embodies the submission format for authors using Word. In Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY. ACM, New York, NY, USA, 10 pages. NOTE: This block will be automatically generated when manuscripts are processed after acceptance.

1 INTRODUCTION

Programming tasks such as fixing a bug, code comprehension, reviewing code, and code summarization requires skill at reading source code. However, researchers have shown that programmers' code reading skills reflect their expertise level [Abid et al. 2019b] [Busjahn et al. 2011]. Thus, studying programmers' reading

behavior provides a practical method that can be used to characterize the differences between experts and novices in a realistic environment.

The study of eye movements is gaining popularity in different software engineering domains. One of the first studies to use eye tracking is [Crosby and Stelovsky 1990] and studies the impact of experience and the viewing strategies when reading a Pascal 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]. They found that the more experienced programmers tend to focus more on the important area of the code and use beacons to comprehend the program more so than novices.

Several studies have also been performed to capture the impact of the expertise level on the navigation strategies and processes that were applied by the programmers. In a programing comprehension domain, Turner et al. compared 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.

Busjahn et al. conducted a study of programmers during a program 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. 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.

In the study presented here, we examine the difference 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.

In 2019, a research study conducted by Abid et al. shows the role of expertise in code summarization tasks [Abid et al. 2019a]. 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. In this paper, we provide an in-depth analysis of the reading process including 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 (as done in [Peterson et al. 2019]).

The contributions of the added analysis in this research are as follows:

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

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] [Siegmund et al. 2014]. Researchers have shown that expertise reflects the performance on how programmers perform and not necessarily how long they have trained [Shanteau 1992]. Therefore, 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. To address the research goals, we seek to answer the following research questions:

- 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)?

2 BACKGROUND

Eye tracking in software engineering [Sharafi et al. 2020] 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 and Maletic 2010b], model comprehension [Sharif 2011] [Sharif and Maletic 2010a], debugging, and code summarization [Rodeghero et al. 2014] [Rodeghero and McMillan 2015]. The following summarizes and demonstrates the findings of several eye tracking studies.

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 [Sharif et al. 2012]. 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.

To facilitate programmer work and to help them avoid introducing bugs into the code, Fritz et al. introduces a new approach to determine task difficulty by using an (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#. 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.

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. 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.

On the use of eye tracking in debugging, 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.

None of the prior work looks at the detail of source code constructs used at the detail presented in this paper.

3 METHODOLOGY

The overall goal of this research is to determine how programmers (experts and novices) read Java source code at the source code construct level. Novices are known to cover more parts of the code than experts [Busjahn et al. 2015], but what exactly do they focus on mostly at the line, statement and term level when reading the programming code? This section describes the process of the study to answer the research questions.

3.1 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]. 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.

3.2 Data Cleaning and Transformation

3.2.1 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

3.2.2 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.

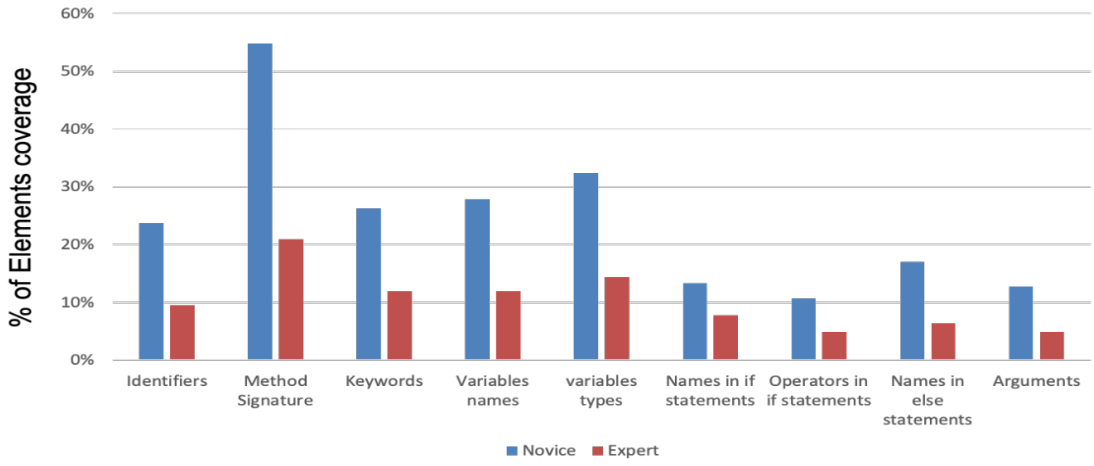

Figure 1: Comparing experts and novices eye movements in reading source code elements

3.2.3 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

3.2.4 Identify Term Location and Mapping:

To map the eye gaze onto the identifying source code elements, first we extract all the navigation areas that we want to study from the code. However, we use srcML [Collard et al. 2011] to generate individual files for each navigation part which provides an XML representation of the code. Then we convert these files to CSV lists. Each list (such as the identifiers 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).

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.

4 RESULTS AND DISCUSSION

***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**?*

For the navigation areas investigated in the study, we found statistical evidence that novices read more source code elements than experts. In Figure 1, 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 1 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 1 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%.

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 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. Effect size is computed using Cohen's d.

Table 1: Mann-Whitney results for comparing novices and experts in navigation areas. Asterisks indicate significant with 95% confidence.

Source code elements	U	p
Identifiers	65	0.027 *
Method signatures	13.5	0.02 *
Variable names	66	0.023 *
Variable types	64.5	0.033 *
Keywords	68	0.014 *
Arguments	54.5	0.212
Operators	59.5	0.091
Names in if statements	55	0.196
Operators in if statements	55	0.197
Names in else statements	65	0.023 *
Operators in else statements	61.5	0.024 *
Names in while statements	61.5	0.055
Operators in while statements	56.5	0.154

The results in Table 1 show that novices have a significantly higher average for reading identifiers in the source code. The difference is statistically significant: $U = 65$, $p = 0.027$ with a large effect size of 1.15. The same holds true when comparing novice programmers with experts for reading method signatures; the difference between the two groups is statistically significant: $U = 13.5$, $p = 0.02$ Cohen's d = 1.12.

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: $U = 68$, $p = 0.014$ with a large effect size of 1.36.

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 ($U = 61.5$, $p = 0.023$ Cohen's d = 1.29). The results also show that the difference is significant between experts and novices when looking at operators in else statements, $U = 61.5$, $p = 0.024$ with a large effect size of 1.24. However, the statistic result indicates that the difference between experts and novices for reading names in while statements is not significant: $U = 61.5$, $p = 0.055$ Cohen's d = 1.01.

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.

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**: **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.

In terms of the navigation area that is most visited when programmers review the code, the results show that 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 and 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 et al. 2019b]. 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.

In summary, we consider the task to be crucial when making claims about cause and effect in reading. The task drives the way people read and plays a big role in the analysis and results.

5 CONCLUSION AND FUTURE WORK

The paper analyzes 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. Moreover, programmers both experts and novices find that signatures are the most important parts when scanning code. We confirm previous results on the impact of expertise on reading behaviors, and add a comparison on the source code element level. For future work, we want to replicate the analysis on additional eye tracking data collected from different software tasks other than fixing bugs as this might affect and change reading strategies.

REFERENCES

ABID, N.J., MALETIC, J.I., AND SHARIF, B. 2019a. 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*, ACM, 13:1-13:9.

ABID, N.J., SHARIF, B., DRAGAN, N., ALRASHEED, H., AND MALETIC, J.I. 2019b. Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 384–395.

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.

BEGEL, A. AND VRZAKOVA, H. 2018. Eye movements in code review. *Proceedings of the Workshop on Eye Movements in Programming - EMIP '18*, ACM Press, 1–5.

BUSJAHN, T., BEDNARIK, R., BEGEL, A., ET AL. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. *2015 IEEE 23rd International Conference on Program Comprehension*, 255–265.

BUSJAHN, T., SCHULTE, C., AND 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*, ACM Press, 1.

COLLARD, M.L., DECKER, M.J., AND 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*, IEEE, 173–184.

CROSBY, M.E., SCHOLTZ, J., AND WIEDENBECK, S. 2002. The Roles Beacons Play in Comprehension for Novice and Expert Programmers. 10.

CROSBY, M.E. AND STELOVSKY, J. 1990. How do we read algorithms? A case study. *Computer 23*, 1, 25–35.

FEIGENSPAN, J., KÄSTNER, C., LIEBIG, J., APEL, S., AND HANENBERG, S. 2012. Measuring programming experience. *2012 20th IEEE International Conference on Program Comprehension (ICPC)*, 73–82.

FRITZ, T., BEGEL, A., MÜLLER, S.C., YIGIT-ELLIOTT, S., AND ZÜGER, M. 2014. Using Psycho-physiological Measures to Assess Task Difficulty in Software Development. *Proceedings of the 36th International Conference on Software Engineering*, ACM, 402–413.

HOLLANDER, M., WOLFE, D.A., AND CHICKEN, E. 2013. *Nonparametric Statistical Methods*. John Wiley & Sons.

KEVIC, K., WALTERS, B.M., SHAFFER, T.R., SHARIF, B., SHEPHERD, D.C., AND FRITZ, T. 2015. Tracing Software Developers' Eyes and Interactions for Change Tasks. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ACM, 202–213.

KEVIC, K., WALTERS, B.M., SHAFFER, T.R., SHARIF, B., SHEPHERD, D.C., AND FRITZ, T. 2017. Eye gaze and interaction contexts for change tasks – Observations and potential. *Journal of Systems and Software 128*, 252–266.

PETERSON, C.S., ABID, N.J., BRYANT, C.A., MALETIC, J.I., AND 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*, Association for Computing Machinery, 1–4.

RODEGHERO, P. AND 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.

RODEGHERO, P., MCMILLAN, C., MCBURNEY, P.W., BOSCH, N., AND 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*, ACM Press, 390–401.

SHAFFER, T.R., WISE, J.L., WALTERS, B.M., MÜLLER, S.C., FALCONE, M., AND 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*, ACM, 954–957.

SHANTEAU, J. 1992. Competence in experts: The role of task characteristics. *Organizational Behavior and Human Decision Processes 53*, 2, 252–266.

SHARAFI, Z., SHARIF, B., GUÉHÉNEUC, Y.-G., BEGEL, A., BEDNARIK, R., AND CROSBY, M. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering 25*, 5, 3128–3174.

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.

SHARIF, B., FALCONE, M., AND 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*, ACM, 381–384.

SHARIF, B. AND MALETIC, J.I. 2010a. The Effects of Layout on Detecting the Role of Design Patterns. *2010 23rd IEEE Conference on Software Engineering Education and Training*, 41–48.

SHARIF, B. AND MALETIC, J.I. 2010b. An Eye Tracking Study on camelCase and under_score Identifier Styles. *2010 IEEE 18th International Conference on Program Comprehension*, 196–205.

SIEGMUND, J., KÄSTNER, C., LIEBIG, J., APEL, S., AND HANENBERG, S. 2014. Measuring and modeling programming experience. *Empirical Software Engineering 19*, 5, 1299–1334.

TURNER, R., FALCONE, M., SHARIF, B., AND 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*, ACM, 231–234.

UWANO, H., NAKAMURA, M., MONDEN, A., AND 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*, ACM Press, 133.