An Eye-tracking Study on the Role of Scan Time in Finding Source Code Defects

Bonita Sharif, Michael Falcone
Dept. of Computer Science and Information Systems
Youngstown State University
Youngstown, Ohio 44555
bsharif@csis.ysu.edu, mrfalcone@student.ysu.edu

Jonathan I. Maletic
Department of Computer Science
Kent State University
Kent, Ohio 44242
jmaletic@kent.edu

Abstract

An eye-tracking study is presented that investigates how individuals find defects in source code. This work partially replicates a previous eye-tracking study by Uwano et al. [2006]. In the Uwano study, eye movements are used to characterize the performance of individuals in reviewing source code. Their analysis showed that subjects who did not spend enough time initially scanning the code tend to take more time finding defects. The study here follows a similar setup with added eye-tracking measures and analyses on effectiveness and efficiency of finding defects with respect to eye gaze. The subject pool is larger and is comprised of a varied skill level. Results indicate that scanning significantly correlates with defect detection time as well as visual effort on relevant defect lines. Results of the study are compared and contrasted to the Uwano study.

CR Categories: H.1.2 [Models and Principles]: User/Machine Systems – Human Factors; D.2.5 [Software Engineering]: Testing and Debugging – Code inspections and walk-throughs.

Keywords: eye-tracking study, source-code review, software defects

1 Introduction

It is well known that code reviews improve the overall quality of software and reduce defects. There are many code reviewing techniques based on checklists, perspectives, defects, usage scenarios, as well as ad-hoc methods proposed in the literature and textbooks. Code reviewing techniques [Ciolkowski et al. 2002] can also be taught as part of the undergraduate computer science or software engineering curriculum. Several empirical studies have shown no conclusive evidence as to which of the above code reviewing methods finds the most defects [Abdelnabi et al. 2004]. One reason is that the performance of individual subjects is more dominant than the review method itself [Uwano et al. 2006].

Due to the observed differences in individuals, Uwano et al. [2006] conduct an eye-tracking study to evaluate individual performance of source code reviews, rather than testing if one review technique outperforms another (e.g., check-list vs. perspective-based). A particular pattern was identified that they called a *scan* in eye gaze data. In short, the scan is the time subjects spent reading over the code initially. While the Uwano study was not statistically significant due to the small number of participants (i.e., 5), they observed that reviewers who did not spend enough time during the scan tend to take more time in finding defects. In order to shed more light on this matter, we replicate and extend the study presented in Uwano et al. [2006]. The study presented here is conducted with a larger number of participants (N=15) with varied programming experience. The results are compared and contrasted to the Uwano study.

The research here focuses on how a software developer reads source code while given a specific task. The task here is reading and comprehension of source code with the goal of finding logical defects. We investigate if the scan pattern holds in our study and the role subject's experience has on the effectiveness and efficiency of the code reviewing process. The end goal as Uwano et al. [2006] point out is to find an efficient way to allow a reviewer to find the maximum number of defects. The results of both studies have direct impact on the education aspect of teaching code review procedures in the classroom [Hundhausen et al. 2009]. It also gives feedback to industry about what types of strategies work in code review tasks.

The remainder of the paper is organized as follows. Section 2 provides the study focus, background information on the scan pattern and research questions. The experiment design is given in Section 3 followed by the results and analyses in Section 4 and discussion in Section 5. We conclude the paper with related work, conclusions and future work.

2 Focus of the Study

The goal of this study is to analyze human subjects' eye gaze data while they perform the task of finding defects in source code based on a given program specification. In the Uwano study the *scan* pattern is defined as follows. *The scan pattern is a preliminary reading process where the reviewer reads the entire code before focusing on a smaller subset of lines.* That is, the time during which a subject initially scans the code to get a basic understanding. An example of a scan pattern is shown in Figure 1. The main research questions this paper addresses are:

- RQ1: Does scan time affect the defect detection accuracy, defect detection time, and visual effort in finding defects in source code?
- RQ2: How does the subject's experience affect the code review process?

The first research question was formulated based on the observation presented by Uwano et al. [2006]. The second research question seeks to determine if there is a significant difference between the eye movements of experienced programmers and novices for code review tasks. It has been shown that experienced programmers tend to focus on key beacons [Brooks 1983] in source code while they are trying to comprehend it. The study seeks to determine if the same findings apply to the code review task.

3 Experimental Design

The experiment seeks to analyze an ad-hoc code reviewing process for the purpose of evaluating individual differences with respect to effectiveness, efficiency, and visual effort from the point of view of the researcher. The study was conducted using the Tobii 1750 eye tracker (www.tobii.com). It is a video-based remote eye tracker that uses two cameras to capture eye gaze. The cameras are built into a 17 inch LCD screen. The screen resolution was set to 1024 by 768. No head gear is necessary making the work environment very similar to normal working conditions. The eye-tracker sampled 50 frames per second with a latency of approximately 25-35 ms, and average accuracy is 0.5 degrees (1.5 pixels of error). Fixation area was determined at 20 pixels in diameter and at least 40 ms in duration.

3.1 Variables and Hypotheses

The dependent variables are Scan Time (time for the first scan pattern to take place), Accuracy, Defect Detection Time, and Visual Effort. Visual Effort is measured by several separate eye-tracking based variables namely, total fixation counts and durations, and relevant fixation counts and durations (fixations and durations on defect line(s)). A secondary explanatory variable is the Experience of subjects with two values: expert and novice. The high-level hypothesis is that the quality of the first scan time significantly influences the individual effectiveness, efficiency, and visual effort of defect detection. The individual hypotheses are given below.

- $H1_0$: Defect detection time is the same regardless of Scan Time
- $H2_0$: Defect detection Accuracy is the same regardless of Scan Time
- $H3_0$: Visual effort is the same regardless of Scan Time
- $H4_0$: Experience does not interact with Scan Time to have an effect on defect detection time and accuracy

3.2 Participants

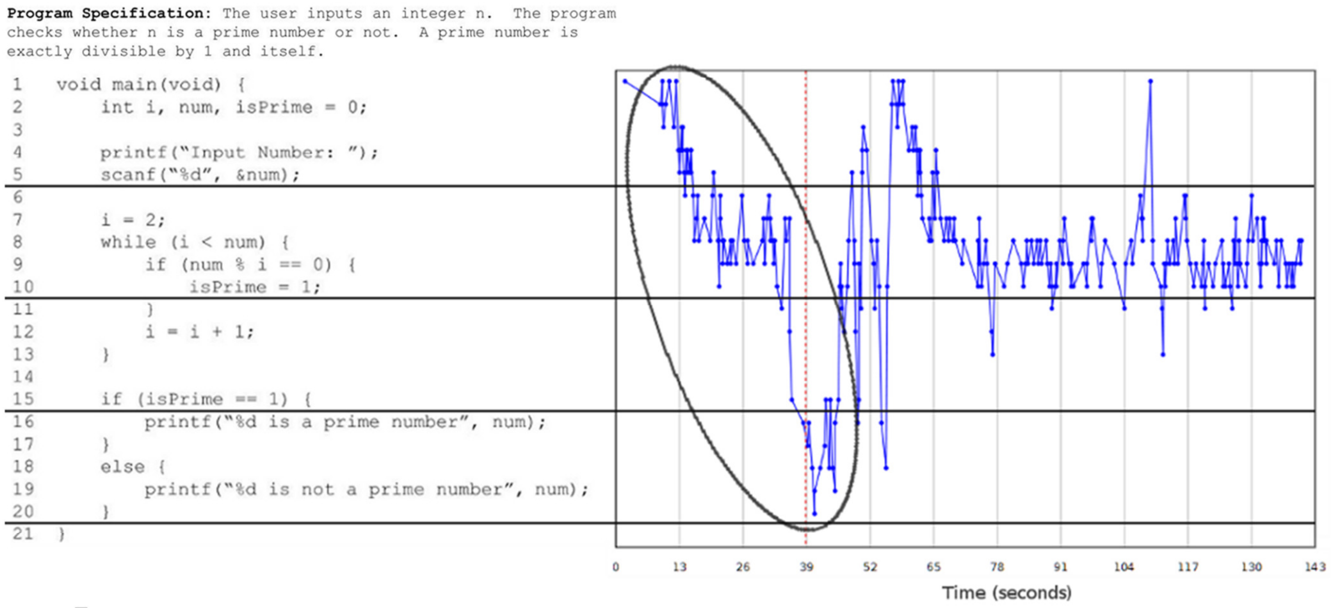
Fifteen volunteers from the Department of Computer Science at Kent State University participated in this study. There were seven undergraduates in their second year of study, six graduate students, and two faculty members. Based on a background pre-questionnaire that gathered demographic data, we classified the participant pool into experts and novice programmers. The expertise assignment was mainly based on the number of years actively programming in both academia and industry. Eight participants were classified as expert and seven were classified as novice. One expert was not analyzed due to insufficient fixation data. Two of the subjects were female. All subjects had normal vision. Some wore contact or corrective lenses. The participants were not aware of the experiment's hypotheses.

Table 1. Programs reviewed

Program	Lines of Code	Defect Description	Defect lines
Sum-5	14	Sum is not initialized	3
Accumulate	22	Boundary condition <=	6, 7
Average-5	16	Type conversion missing	14
Prime	21	Logic is backwards	2, 10

3.3 Stimuli, Tasks, and Injected Defects

The main task was to find the defect in four source code snippets. The source code snippets were the same as those used in the Uwano study [2006]. Each program had one logical defect with no syntax errors. See Table 1 for summary of the programs used. Sum-5 calculates the sum of five integers. Accumulate outputs the sum of integers from 1 to n. Average-5 outputs the average of five integers and finally Prime checks if a user input n is prime. The stimulus given to the subject is shown on the left side of Figure 1. The program specification is given on top. The subject was instructed to read the specification before trying to find the defect. They were allowed to ask questions on the specifications and the C language during the review similar to the Uwano study. Before the study began, the participants' eyes were first calibrated. Next, they were briefed on the objective of the study. The participants were instructed to think aloud and verbally state the line number(s) they thought the defect was at. If they were unable to find a defect, they moved to the next stimulus.



Program Specification: The user inputs an integer n. The program checks whether n is a prime number or not. A prime number is exactly divisible by 1 and itself.

```
1   void main(void) {
2       int i, num, isPrime = 0;
3
4       printf("Input Number: ");
5       scanf("%d", &num);
6
7       i = 2;
8       while (i < num) {
9           if (num % i == 0) {
10              isPrime = 1;
11          }
12          i = i + 1;
13      }
14
15      if (isPrime == 1) {
16          printf("%d is a prime number", num);
17      }
18      else {
19          printf("%d is not a prime number", num);
20      }
21  }
```

Figure 1. Subject 5 reviewing Prime. The scan pattern is evident in the first 39 seconds where the subject reads the entire code before concentrating on the while loop. The red dotted line represents the time to first scan. The left side represents the stimulus that was shown to the user. The right side is fixations in time generated using our eye gaze analyzer written in C++.

4 Experimental Results and Analyses

During data collection and analysis, we had to discard data for one subject leaving us with 56 trials instead of 60. The eye movement patterns were visualized using our eye gaze analyzer written in C++. In order to visualize groups (experts vs. novices), we graphed average eye movements for each stimulus for both the group of experts and the group of novices. The eye tracking device logs each fixation when a fixation change is detected, rather than logging the location of the current fixation at a constant rate. Because of this, we sampled the fixations from the eye tracking device's logs at a constant rate so that each session has the same number of samples. If a sampled fixation did not fall on a line, we used the most recent line for the sample. This allowed us to visualize a continuous graph of line fixations over the entire session. The right side of Figure 1 shows the visualized eye movements while Subject 5 (a novice) was reviewing the Prime program. We analyzed all eye movements using our eye gaze analyzer to identify patterns. See Figure 2 for descriptive statistics on the two dependent variables. They show Prime and Accumulate tend to be harder than Sum-5 and Average-5. There is also a lot less variation in defect detection time in Sum-5 and Average-5 vs. Prime and Accumulate.

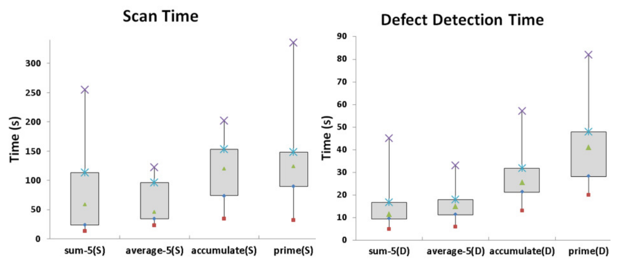


Figure 2. Descriptive Statistics for first Scan Time and Defect Detection Time

Scan Pattern: We first looked at eye movements of each subject, and identified a scan pattern for most individuals. With respect to the scan pattern, we find 73.85 percent (Experts: 70.8 percent, Novices: 76.9 percent) of code lines were watched in the first 30 percent of the review time. These results are consistent with the Uwano study (72.8 percent). Figure 1 very closely resembles Subject E reviewing Prime in the Uwano study [2006].

Characterizing Review Performance by Scan Pattern: The first scan time (as in the Uwano study) is defined as the time spent from the beginning of the review until 80 percent of the total lines are read. The defect detection time is the time taken for a reviewer to detect the injected defect. We analyze the correlation using Spearman rank test between scan time and the dependent variables thereby seeking to validate our hypotheses.

From Table 2, we conclude that Scan time is significantly correlated with Defect detection time thereby rejecting $H1_0$. Scan time is also significantly correlated with total fixation counts, total fixation durations (only for entire dataset of correct and incorrect answers), relevant fixation counts, and relevant fixation durations. So with respect to visual effort we can reject the null hypothesis $H3_0$. The most important measure in visual effort is that of the relevant fixation counts. It tells us the focus of the subject after the first scan. Scan time is not significantly correlated with accuracy for the entire data set. We had 28 correct and 28 incorrect answers. Out of the 28 correct answers, 21 were experts and 7 novices and the opposite applies to the set of incorrect answers. We cannot reject $H2_0$.

Experts vs. Novices: Based on a mixed models linear regression model where the explanatory variables are Scan time and Experience, we find that novices tend to take an average of 25.84 seconds longer in finding the defect (p-value= 0.063), which approaches significance. However, there was a significant difference in the accuracy of the defects between experts and novices (p-value=0.00001). From this we can deduce that experience does interact with Scan Time to have an effect on accuracy and defect detection time thereby rejecting $H4_0$.

Table 2. Spearman results for Scan Time. The values are reported for the entire data set (N=56) as well as for correct answers only in parenthesis (N=28).

Dep.Variable	R_s	**2 tailed p-value**	**Result**
Defect Detection Time	0.53(0.53)	<0.0001 (0.0034)	Reject $H1_0$
Total Fix. Counts	0.37(0.46)	0.0045 (0.0147)	Reject $H3_0$
Total Fix. Durations	0.32(0.32)	0.01 (0.1022)	Reject $H3_0$ for entire data set. Cannot reject $H3_0$ for dataset with correct answers
Relevant Fix. Counts	0.43(0.62)	0.0009 (0.0004)	Reject $H3_0$
Relevant Fix. Dur.	0.31(0.43)	0.0191 (0.0217)	Reject $H3_0$
Accuracy (for Entire Data Set)	-0.21(-)	0.1124(-)	Cannot Reject $H2_0$

To qualitatively compare scan times between the two groups (expert vs. novice), we defined the scan time for the group to be the average of the scan times of each subject in that group. For each stimulus, the expert group had a lower scan time than the novice group. To visualize performance, we used a scatter plot for scan time against total time for each subject across all stimuli. See Figure 3. The horizontal axis shows the time taken to complete the first scan and the vertical axis shows the total time taken to find the defect. Each axis is normalized by the average of the values represented by that axis. The scatter plots show that for both experts and novices a longer scan time tends to reduce overall time to find the defect. The figure shows that first scan time less than the average (1.0) yields longer defect detection times. Specifically, the defect detection time increased up to 3.5 times of average detection time when first scan time is less than 1.0. On the other hand, if scanning time is more than 0.8, the defect detection time is less than the average.

We averaged the eye movements over each group (expert vs. novice) using the arithmetic mean and graphed the results. In the averaged graphs (not shown here) we noticed several patterns with the scan shape. For each of the four stimuli, the standard deviation of the fixation line expert graph was lower than the novice graph.

Threats to validity: Due to the way individual eye movements are sampled prior to calculating the group average, there is some loss of fixation data. Care was taken to choose an appropriate sampling rate to visualize average movements, and our results seem to be consistent among different sampling rates.

Comparison to the Uwano Study: In the Uwano et al. [2006] study, five graduate students participated in the experiment with three or four years programming experience. In our experiment, we employ a total of 15 subjects (undergraduates, graduates, and faculty) with varied expertise and years in programming. We used only four of the six C programs used in the Uwano study. The text of the programs was exactly the same. A different eye tracker (Tobii 1750) was used to collect the data. A slight difference exists in study instrumentation where explanation of the specification was not done verbally to the subject, instead the subject read the specification on the screen above the actual code. We did not wait until a subject finds the correct defect or total time exceeds 5 minutes. As soon as the subject gave an answer, we went to the next task but they were encouraged to try their best. The Uwano experiment does not report significant correlation values due to low sample size. Interviews were not conducted in our study. Our work highlights the differences and similarities of fixation patterns among experts and novices as two separate groups rather than looking at each individual performance.

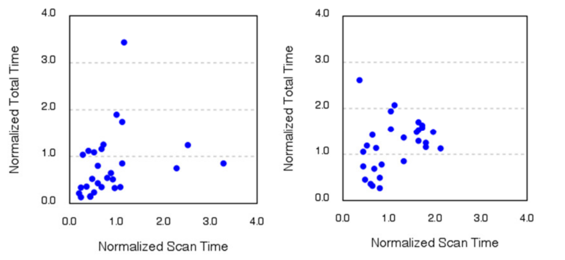


Figure 3. Correlation between first Scan Time and Defect Detection Time for experts(left) and novices (right).

5 Related Work

Although there are many eye-tracking studies on evaluating user interfaces, there are very few studies done by a handful of researchers on how programmers read and comprehend source code. Crosby and Stelovsky studied the eye gaze behavior of nineteen novices and experts to determine if experience changed eye gaze patterns [Crosby and Stelovsky 1990]. At about the same time as the Uwano study [2006], Bednarik et al. also studied the eye movement of eighteen subjects comprehending Java programs using a program visualization tool [Bednarik and Tukiainen 2006]. Later on, they study the eye gaze of pair programmers simultaneously [Pietinen et al. 2008]. Uwano et al. also study reviewer eye-movements in multi-view documents [Uwano et al. 2008]. Bednarik et al. also investigate debugging behavior of fourteen subjects in an IDE setting. [Bednarik and Tukiainen 2008]. Recently, Sharif et al. study the impact of identifier style (i.e., camel case or underscore) on code reading and comprehension using an eye-tracker [Sharif and Maletic 2010].

6 Discussion and Conclusions

The paper replicates an eye-tracking source code review study by Uwano et al. [2006]. A larger number of participants as well as statistically significant results along with additional eye tracking measures add to the validation of the previous study. The results indicate that the longer a reviewer spends in the initial scan, the quicker they find the defect. Conversely, if a reviewer does not spend sufficient time on the scan they are likely to find irrelevant lines thereby decreasing the performance of the review. The scan time plays an important role in defect detection time and visual effort needed to review source code.

The expert group's standard deviation (St dev.=1.5) compared to the novice group's (St dev.=2.03) suggests that experts tend to focus on fixations more, while novices may watch lines more broadly. A possible explanation of this is that experts may associate certain lines with potential likely problems, based on internalized standard programming plans [Brooks 1983], and focus attention to those lines, while novices may have to follow code more closely in order to recognize where defects might occur. In all stimuli, novices spent more time with scan than experts. This may again be due to an intuitive notion of problem areas possessed by experts. There are cases where the subject stated the correct line number but not necessarily the correct defect on that line. For example, for the Average-5 program one subject stated the defect on line 14 but instead of stating that there will be truncation on average, they stated incorrectly that variable i should be 5 instead of 4. In the Uwano study [2006], they found some patterns of retracing where a reviewer often looks back to the declaration of the variable. We did not notice a significant amount of retrace patterns in this study.

Since the eye movements between experts and novices follow different patterns, it is possible to use eye-tracking measures to determine the skills of a developer in a specific software engineering task. More studies need to be done to support this conjecture. As future work, it would be interesting to identify how scan patterns differ among control structures and different program lengths. It may also be interesting to see which lines experts and novices are more likely to fixate on, i.e. control structure statements, arithmetic operations, and/or logical expressions. More experiments that mimic real-life coding environments are needed to answer the above questions.

References

ABDELNABI, Z., CANTONE, G., CIOLKOWSKI, M., and ROMBACH, D. 2004. Comparing Code Reading Techniques Applied to Object-Oriented Software Frameworks with regard to Effectiveness and Defect Detection Rate. in *Proc. of Intl. Symposium on Empirical Software Engineering*, pp. 239 - 248

BEDNARIK, R. and TUKIAINEN, M. 2006. An Eye-tracking Methodology for Characterizing Program Comprehension Processes. in *Proceedings of Symposium on Eye tracking research & Applications (ETRA)*, San Diego, pp. 125-132.

BEDNARIK, R. and TUKIAINEN, M. 2008. Temporal Eye-tracking Data: Evolution of Debugging Strategies with Multiple Representations. in *Proceedings of Symposium on Eye Tracking Research & Applications (ETRA)*, Savannah, pp. 99-102.

BROOKS, R. 1983. Towards a Theory of the Comprehension of Computer Programs. *International Journal of Man-Machine Studies*, vol. 18, no. 6, pp. 543-554.

CIOLKOWSKI, M., LAITENBERGER, O. R., D., SHULL, F., and PERRY, D. 2002. Software Inspections, Reviews & Walkthroughs. in *Proceedings of International Conference on Software Engineering (ICSE)*, pp. 641-642.

CROSBY, M. E. and STELOVSKY, J. 1990. How do we read algorithms? A case study. *IEEE Computer*, v. 23, n. 1, pp. 24-35.

HUNDHAUSEN, C., AGRAWAL, A., FAIRBROTHER, D., and TREVISAN, M. 2009. Integrating Pedagogical Code Reviews into a CS 1 Course: An Empirical Study. in *Proc. of SIGCSE*, Chattanooga, Tennessee, pp. 291-295.

PIETINEN, S., BEDNARIK, R., GLOTOVA, T., TENHUNEN, V., and TUKIAINEN, M. 2008. A Method to Study Visual Attention Aspects of Collaboration: Eye-tracking Pair Programmers Simultaneously. in *Proceedings of Symposium on Eye Tracking Research & Applications*, Savannah, Georgia, pp. 39-42.

SHARIF, B. and MALETIC, J. I. 2010. An Eye tracking Study on CamelCase and Under_score Identifier Styles. in *Proceedings of 18th IEEE Intl. Conf. on Program Comprehension*, Braga, Jun 30-Jul 2, pp. 196-205.

UWANO, H., MONDEN, A., and MATSUMOTO, K.-I. 2008. DRESREM 2: An Analysis System for Multi-document Software Review Using Reviewers' Eye Movements. in *Proc. of Intl. Conf. on Software Engineering Advances*, Malta, pp. 177 - 183

UWANO, H., NAKAMURA, M., MONDEN, A., and MATSUMOTO, K. 2006. Analyzing individual performance of source code review using reviewers' eye movement. in *Proc.of Symp. on Eye tracking research&applications*, San Diego, pp.133-140