Factors Influencing Dwell Time During Source Code Reading - A Large-Scale Replication Experiment

Cole S. Peterson
University of Nebraska - Lincoln
Dept. of Computer Science and Eng.
Lincoln, Nebraska, USA 68588
cole.scott.peterson@huskers.unl.edu

Nahla J. Abid
Taibah University
Department of Computer Science
Madinah, Kingdom of Saudi Arabia
nabd@taibahu.edu.sa

Corey A. Bryant
Kent State University
Department of Computer Science
Kent. Ohio, USA 44240
cbryan20@kent.edu

Jonathan I. Maletic
Kent State University
Department of Computer Science
Kent. Ohio, USA 44240
jmaletic@kent.edu

Bonita Sharif
University of Nebraska - Lincoln
Dept. of Computer Science and Eng.
Lincoln, Nebraska, USA 68588
bsharif@unl.edu

ABSTRACT

The paper partially replicates and extends a previous study by Busjahn et al. [4] on the factors influencing dwell time during source code reading, where source code element type and frequency of gaze visits are studied as factors. Unlike the previous study, this study focuses on analyzing eye movement data in large open source Java projects. Five experts and thirteen novices participated in the study where the main task is to summarize methods. The results examine semantic line-level information that developers view during summarization. We find no correlation between the line length and the total duration of time spent looking on the line even though it exists between a token's length and the total fixation time on the token reported in prior work. The first fixations inside a method are more likely to be on a method's signature, a variable declaration, or an assignment compared to the other fixations inside a method. In addition, it is found that smaller methods tend to have shorter overall fixation duration for the entire method, but have significantly longer duration per line in the method. The analysis provides insights into how source code's unique characteristics can help in building more robust methods for analyzing eye movements in source code and overall in building theories to support program comprehension on realistic tasks.

CCS CONCEPTS

• **Human-centered computing** → **Empirical studies in HCI**; • **Software and its engineering** → *General programming languages*;

KEYWORDS

source code, eye tracking study, visual attention distribution

ETRA '19, June 25–28, 2019, Denver, CO, USA

ACM ISBN 978-1-4503-6709-7/19/06…$15.00
https://doi.org/10.1145/3314111.3319833

ACM Reference Format:
Cole S. Peterson, Nahla J. Abid, Corey A. Bryant, Jonathan I. Maletic, and Bonita Sharif. 2019. Factors Influencing Dwell Time During Source Code Reading - A Large-Scale Replication Experiment. In *2019 Symposium on Eye Tracking Research and Applications (ETRA '19), June 25–28, 2019, Denver, CO, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3314111.3319833

1 INTRODUCTION

Developing software is a complex mental activity that can be broken down into many sub-tasks. One such task is that of reading source code. In recent work, researchers have shown that source code reading is indeed different from natural language reading [Busjahn et al. 2015]. The strategies developers use to read source code are important because it greatly influences the time taken to solve a task such as fixing a bug or implementing a new feature. Program comprehension [Brooks 1983; Soloway and Ehrlich 1989] is a sub-area of software engineering that focuses on how developers read and understand source code. Most of the eye tracking studies done in program comprehension however, focus on small source code snippets. We direct the reader to prior systematic literature reviews [Obaidellah et al. 2018; Sharafi et al. 2015] for more details.

In 2014, Busjahn et al. conducted an eye tracking study in which they analyzed dwell time on the source code token's syntactic category, length, and frequency [Busjahn et al. 2014]. They generated a set of 11 small Java programs as the stimuli. They found that most attention is spent on identifiers, operators, keywords, and literals. However, beyond this work there are no studies done on large source code stimuli to determine how developers distribute their attention across the different semantic lines within source code. In order for such results to be meaningful and impact realistic software tools that use this data, it is important to determine if such findings hold when more realistically sized programs are used. In this paper, *we partially replicate the analysis done in the Busjahn et al. study using an eye tracking dataset [Abid et al. 2019] that was collected in a realistic setting.* The main difference between this study and the Busjahn et al. study is that the data is collected on large open source software with the methods being surrounded by the original source code in its file and was done within the developer work environment (i.e., Eclipse IDE). The open source infrastructure iTrace [Guarnera et al. 2018] is used to gather the data. Another difference

is that instead of focusing purely on the syntactic tokens in the source code, we investigated the visual attention and distribution based on the semantic meaning of the token and surrounding line e.g., an if-statement or assignment-statement. We found that several trends presented in Busjahn et al. are not present at the line level analysis in our dataset.

2 STUDY OVERVIEW

We analyze an existing dataset that consists of data from 18 participants. Five of these are considered experts (more than 5 years of experience in programming) and the remaining 13 are novices (approx. a year of programming experience). The participants are randomly assigned 15 methods to summarize out of the total number of 63 methods chosen from five Java projects. The methods to be summarized ranged from 9 to 80 lines of code with a median of 22 LOC and a mean of 27.07 LOC. The method's cyclomatic complexity ranged from 1 to 28 with a median of 6 and a mean of 7.16. The main task of the study is to read the assigned methods and write a summary for them in a text file.

During the summarization task, the participants are seated in front of a 24-inch LCD monitor and their eye movements and fixations are recorded using the Tobii X60 eye tracker along with iTrace [Guarnera et al. 2018] to map these fixations to the correct line and column number in the source code regardless of scrolling or code folding. The screen resolution was 1920 px × 1080 px. A more extensive explanation of the experimental set up can be found in [Abid et al. 2019]. The source code and all methods to be summarized can be found in our replication package at http://seresl.unl.edu/ETRA2019.

3 STUDY RESULTS

We use the dwell time to determine visual attention as we want to compare our results to [Busjahn et al. 2014]. Dwell time is the sum of all fixation durations during a single visit of an AOI. We find that the dwell time distribution is right-skewed just as in the Busjahn study. In order to transform this into a normal distribution, the dwell time distribution is log-transformed. This distribution is similar to what is reported in [Busjahn et al. 2014].

3.1 Dwell Time Compared to Line Length

As shown in [Busjahn et al. 2015], an element with more characters takes longer to read. Due to this effect, normalization of the dwell time must occur based on the element's length. However, because we are investigating line based trends, the goal is to determine if this effect is present with the line's length and its corresponding dwell time. Using the log-transformed dwell times found in the previous section, we attempted to find a correlation between the length of the line and the total dwell time seen in Figure 1. The linear model found for this correlation does not show *any* significant relationship between the two variables (R^2=0.007011). We conclude that line length has no statistically significant effect on the total dwell time of the line. We also investigated if this trend occurs in any of the types of lines that categorized. While some line types, such as *For* and *If* statement lines, have stronger correlations (R^2=0.0773 and R^2=0.06081 respectively), we were unable to find any significant correlations between line length and total dwell time.

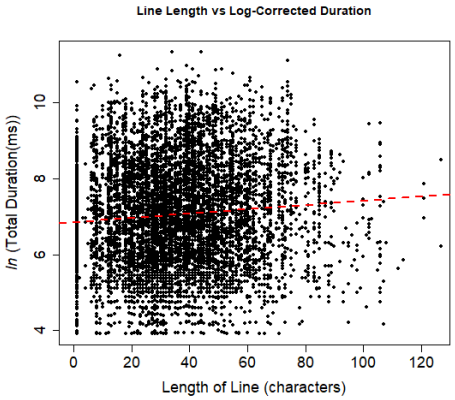

Figure 1: Dwell time over Line Length

3.2 Distribution of Dwell Time over Line Type

The time spent looking at several different types of lines is examined. The program is broken down into 9 different types of semantic meaning. *Method Call*, *If*, *Variable*, *Method Signature*, *For*, *Import*, *Class Attribute*, and *Class* are the different semantic meanings of the lines in our analysis. *Method Call* is used to describe a line that contains a call to another function inside the class or a method from an object. *If* is used to describe a line containing an if or else keyword or the corresponding conditional statement. *Variable* is used to describe a line that contains a local variable declaration or variable assignment. *Method Signature* is used to describe a line that contains the definition of a method along with its parameters. *For* is used to describe the for or while keyword or the corresponding loop conditions. *Import* is used to describe a line with an import statement. *Class Attribute* is used to describe a line that contains a class attribute declaration inside the class declaration. *Class* is used to describe a line that contains the class declaration and the inherited classes and interfaces.

When we break down the dwell times of our subjects into these 9 semantic line types, the following distribution of dwell time is found (Table 1). The three most common line types based on the total dwell time of our subjects are *Method Call*, *If*, and *Variable*. This is consistent when the distribution is based on the frequency of visits to a line type. We also observe that the lowest line types are *Import*, *Class Attribute*, and *Class*. As these line types are infrequent in the code base as can be seen in Figure 2 and non-existent in the methods the subjects are asked to summarize, it makes sense to see these line types at the bottom of the distribution.

One study [Rodeghero et al. 2014] that looked at similar distributions found that *Method Signatures* are the most important type of element for comprehension and code summarization tasks while we found that *Method Signatures* only accounted for 6.1% of the total dwell time of our subjects and other line types accounted for a larger percentage of the total dwell time. However, this previous study [Rodeghero et al. 2014] used much shorter code snippets for their summarization tasks than the dataset we used, so this difference is to be expected. The point to note is that context around the method to be summarized matters.

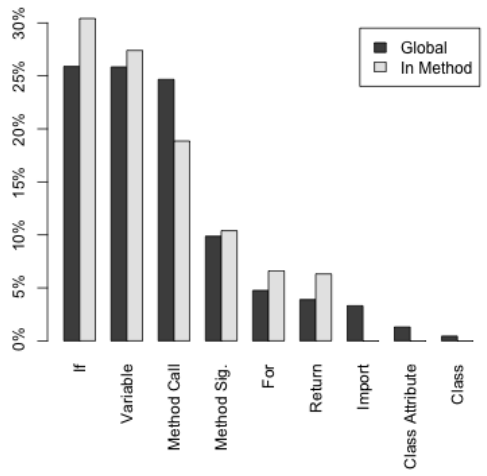

Figure 2: Distribution of line types in terms of percentage of total lines both globally and in the summarized methods

Table 1: Distribution of dwell time as percentage of total frequency with percentage of total duration reported in parentheses

Line Type	All Subjects	Expert	Novice
Class	0.61 (0.76)	0.31 (0.64)	0.69 (0.79)
Class Attr.	0.89 (0.88)	0.41 (0.52)	1.02 (0.97)
Import	2.55 (3.11)	2.89 (3.54)	2.45 (3.00)
For	3.62 (3.35)	3.55 (2.96)	3.64 (3.45)
Method Sig.	5.24 (6.10)	4.48 (6.12)	5.46 (6.09)
Return	7.55 (8.30)	4.91 (5.79)	8.30 (8.97)
Variable	17.90 (16.83)	17.81 (17.61)	17.93 (16.62)
If	23.43 (23.56)	25.08 (23.98)	22.96 (23.45)
Method Call	38.22 (37.11)	40.56 (38.84)	37.55 (36.65)

Table 2: Distribution of dwell time inside summarized methods as percentage of total frequency with percentage of total duration reported in parentheses

Line Type	All Subjects	Expert	Novice
Method Sig.	3.21 (3.96)	3.68 (4.75)	3.06 (3.74)
For	3.84 (3.62)	3.72 (3.16)	3.88 (3.75)
Return	7.66 (8.57)	4.59 (5.46)	8.60 (9.43)
Variable	19.07 (18.14)	18.68 (18.66)	19.18 (17.99)
If	25.39 (25.59)	26.69 (26.27)	25.00 (25.40)
Method Call	40.82 (40.13)	42.63 (41.71)	40.28 (39.69)

3.3 Comparison of Experts and Novices

In an attempt to find any significant differences between expert and novice reading time during code summarization tasks, we examine the dwell time distributions of both experts and novices over the 9 line types defined previously. The two distributions are almost the same. To verify this, we run the Mann-Whitney test at 95% confidence due to the non-parametric (unpaired) nature of the data. The results are not significant based on dwell time (p=0.07 Cohen's d=0.90) and on frequency (p=0.09 Cohen's d=0.85).

3.4 First Fixation Duration and Line Frequency

In [Busjahn et al. 2014], they investigated a trend found in natural language reading which states that words that are less frequent will have longer first fixation duration and longer first dwell time [Rayner and Duffy 1986]. They found that this trend does not exist between keyword frequency and first fixation duration in Java. We examined if a trend occurs between the number of various line types and the first fixation duration on a given line type. To see this effect at different scopes, we used the line type frequency of all files used in the study, the file the subject is summarizing, and the method the subject is summarizing. For the last trend, we remove any gazes from outside the method to investigate if the trend exists when looking inside the methods being summarized. We failed to find any significant correlation between any of the relationships investigated for the global-level (R^2=0.00040), file-level (R^2= 0.00013), and method-level (R^2=0.000030) frequency. It can be concluded that frequency of semantic line types has no significant effect on the first fixation duration.

Table 3: Distributions of first and last fixations as percentage of total frequency and percentage of total duration in parentheses.

Line Type	First Fixation	Last Fixation
For	2.37 (2.42)	3.95 (3.18)
Return	2.37 (3.82)	11.86 (16.94)
If	19.76 (16.90)	19.76 (26.23)
Variable	24.11 (24.98)	20.95 (16.19)
Method Call	24.51 (19.13)	41.11 (36.05)
Method Sig.	26.88 (32.76)	2.37 (1.40)

3.5 First and Last Fixations Inside Methods

Table 2 shows a distribution of all fixations inside summarized methods. The first fixations inside the method are identified from Table 3 for each subject's code summarization tasks and categorized by line type. As can be seen in Table 3, *Method Signature Method Call*, and *Variable* are the most common line types that are first fixated on inside the method. The second most common is *Method Call*, which is the most common fixation type both in terms of duration and frequency when all fixations are taken into account, only accounts for 24.51% of total duration of first fixations and 19.13% of visits. While *Method Signature* and *Variable* normally only account for 5.24% and 17.90% of all fixations, they account for more in the first fixations. The *Method Signature* is less common to be viewed than *Variable* lines when all fixations are taken into account. This large relative increase in *Method Signature* line types being viewed shows that *Method Signature* is common for developers to look at in the beginning of summarizing a method.

We also examine the last fixation a subject makes inside the method they are summarizing (Table 3). While *Method Signature* makes up more of the fixations both in terms of duration and total visits for the first fixation, we do not see this trend in the last

Table 4: Distribution of fixations in large methods in percentage of total frequency and total dwell time in parenthesis

Line Type	Large Methods	Small Methods
Method Sig.	2.62 (3.18)	4.03 (5.11)
For	4.05 (3.93)	3.55 (3.18)
Return	7.98 (8.53)	7.23 (8.61)
Variable	20.16 (19.69)	17.52 (15.85)
If	28.51 (27.81)	20.98 (22.31)
Method Call	36.68 (36.86)	46.69 (44.93)

fixation. Instead, we find that the last fixation is more similar to the overall distribution of fixations. *Return* lines are slightly more common in the last fixations than the overall fixation distributions, but this difference is not large.

3.6 Small vs. Large Methods

As this is one of the few studies that have access to eye tracking data sets with methods that have a large variance in lines of code, we analyzed the largest and smallest methods that our subjects summarized. We chose to classify small methods as 22 lines of code or less and classify large methods as containing more than 22 lines of codes. We chose this cutoff point because in previous studies, 22 lines of code was the largest size method that could be analyzed and was the median size of the programs we studied[Rodeghero and McMillan 2015] [Rodeghero et al. 2015].

As can be seen in Table 4, the first major change is that in smaller methods participant fixate on *For* statements and *If* statements less and *Method Call* and *Method Signature* lines more. This could be because smaller methods are less likely to have longer statements like *If* or *For* because they usually add several lines to the method. However, after running a Mann-Whitney test we find that the differences in distributions between small and large methods are not statistically significant between either duration (p=0.58) or frequency (p=0.58). We also investigate the total duration spent fixating on lines in the method. It was found that participants spent 80 seconds on average fixating on the large methods and spent 70 seconds on average fixating on the small methods. Since larger methods have more lines of code to look at and summarize than smaller methods, this is to be expected. However, when we instead look at the average time spent in a method based on the lines of code the method contains, it is found that participants looking at larger methods spend on average 1.893 seconds per line in the method while participants looking at shorter methods spend on average 3.708 seconds per line (Mann Whitney U test $p < 0.0001$, medium effect size Cohen's d=0.57).

4 CONCLUSIONS AND FUTURE WORK

The paper reports on factors that influence dwell time during source code reading in realistically sized Java programs where the task is to summarize methods. We found that several trends that occur with the syntax level tokens from prior work [Busjahn et al. 2014], are not present at the semantic line-level. There is no correlation between the line length and the total duration of time spent looking on the line even though it exists between a token's length and the total fixation time on the token reported in prior work. The first fixations in a method are more likely to be on a method's signature, a variable declaration, or an assignment. In addition, we found that smaller methods have shorter overall fixation duration for the entire method, but significantly longer duration per line in the method.

Since source code is syntactically and semantically different from natural language text, many of the methods used to analyze natural language text do not always work on source code (e.g., drift correction). If we are aware of the reading patterns and visual distribution on specific source code elements, we can use this information to derive better heuristics for drift correction specific to source code [Palmer and Sharif 2016]. Replications, such as ours, are important from a scientific standpoint as they use realistic work environments [Guarnera et al. 2018] to collect data which mimic the real world setting of how a developer works. This is the first step in our long-term goal of developing theories in understanding program comprehension based on eye movement data.

ACKNOWLEDGMENTS

We are grateful to all participants who took part in the study. This work is supported in part by grants from the National Science Foundation under grant numbers CCF 18-55756 and CCF 15-53573.

REFERENCES

Nahla Abid, Bonita Sharif, Natalia Dragan, Hend Alrasheed, and Jonathan Maletic. 2019. Developer Reading Behavior while Summarizing Java Methods: Size and Context Matters. In *Proceedings of the 41st International Conference on Software Engineering (ICSE) 2019, Montreal, QC, Canada May 25-31, 2019.* 12 pages to appear.

Ruven Brooks. 1983. Towards a theory of the comprehension of computer programs. In *International Journal of Man-Machine Studies, 1983.* 543–554.

T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension.* 255–265.

Teresa Busjahn, Roman Bednarik, and Carsten Schulte. 2014. What Influences Dwell Time During Source Code Reading?: Analysis of Element Type and Frequency As Factors. In *Proceedings of the Symposium on Eye Tracking Research and Applications (ETRA '14).* ACM, New York, NY, USA, 335–338. https://doi.org/10.1145/2578153.2578211

Drew T. Guarnera, Corey A. Bryant, Ashwin Mishra, Jonathan I. Maletic, and Bonita Sharif. 2018. iTrace: Eye Tracking Infrastructure for Development Environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications (ETRA '18).* Article 105, 3 pages.

Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. 2018. A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Comput. Surv.* 51, 1, Article 5 (Jan. 2018), 58 pages.

Christopher Palmer and Bonita Sharif. 2016. Towards automating fixation correction for source code. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications, ETRA 2016, Charleston, SC, USA, March 14-17, 2016.* 65–68. https://doi.org/10.1145/2857491.2857544

Keith Rayner and Susan A Duffy. 1986. Lexical complexity and fixation times in reading: Effects of word frequency, verb complexity, and lexical ambiguity. *Memory & cognition* 14, 3 (1986), 191–201.

P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan. 2015. An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization. *IEEE Transactions on Software Engineering* 41, 11 (2015), 1038–1054.

P. Rodeghero and C. McMillan. 2015. An Empirical Study on the Patterns of Eye Movement during Summarization Tasks. In *2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, Vol. 00. 1–10.

Paige Rodeghero, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving Automated Source Code Summarization via an Eye-tracking Study of Programmers. In *Proceedings of the 36th International Conference on Software Engineering (ICSE 2014).* 390–401.

Z. Sharafi, Z. Soh, and Y. Guéhéneuc. 2015. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology* 67 (2015), 79 – 107.

E. Soloway and K. Ehrlich. 1989. Software Reusability. ACM, New York, NY, USA, Chapter Empirical Studies of Programming Knowledge, 235–267. https://doi.org/10.1145/75722.75734