An Empirical Study on the Patterns of Eye Movement during Summarization Tasks

Paige Rodeghero* and Collin McMillan*
*Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
{prodeghe, cmc}@nd.edu

***Abstract*—Eye movement patterns are the order in which keywords or sections of keywords are read. These patterns are an important component of how programmers read source code. One strategy for determining how programmers perform summarization tasks is through eye tracking studies. These studies examine where people focus their attention while viewing text or images. In this study, we expand on eye tracking analysis to determine the eye movement patterns of programmers. We begin the study with a qualitative exploration of the eye movement patterns used by 10 professional programmers from an earlier study. We then use what we learned qualitatively to perform a quantitative analysis of those patterns. We found that all ten of the programmers followed nearly identical eye movement patterns. These patterns were analogous to eye movement patterns of reading natural language.**

I. Introduction

Eye movement patterns are the sequences or order of eye movements that people use to read words [1]. These patterns are usually researched with respect to natural language text. Areas of patterns research involving natural language text include ad placement in phone books [2], web site design [3], newspaper layout [1], etc. Reading natural language prose is different enough from reading programming languages that eye movement patterns from natural language cannot necessarily be applied to programming languages [4]. For example, advertisements include pictures and have strict size limitations. Programming languages, in contrast, typically do not have pictures and do not have strong screen size restrictions.

Eye movement patterns are important for understanding summarization task performance because they reveal the order in which programmers look at different elements of code. When reading code, programmers "skim" to avoid reading every line of that code. A strong consensus has formed to confirm that skimming occurs [5], though current literature does not agree on *what* the programmers skip. Studies of the eye movement patterns during summarization tasks are needed to determine what the programmers skip. By providing this knowledge, eye movement patterns would provide useful guidance for improving algorithms and tools used to increase summarization task efficiency, such as IDEs and automatic summarization tools.

Unfortunately, current literature does not describe eye movement patterns of programmers. A number of studies have tracked the eye movements of programmers [6], [7], such as a previous eye tracking study we conducted [8]. Our previous work studies discreet locations of code such as method signatures or control flow sections [8]. However, the patterns of eye movement — the tendencies of movement from one section to another — are still largely unknown. The novelty of this work comes from the analysis and deeper understanding of these patterns and their application. Without knowing these patterns, tools meant to aid programmers in summarization tasks could assume incorrect reading patterns such as those used in natural language reading, which is not necessarily the same as for source code. Tools with incorrect underlying assumptions have the possibility of actually slowing or preventing a programmer's understanding of a program.

In this paper, we present a study to extract programmers' reading patterns when applied to source code. Our contribution is two-fold. We first explore, in a qualitative study, the reading patterns from a previous eye-tracking study of programmers reading Java methods [8]. The purpose of our qualitative study is to provide a basis for designing a follow-up quantitative study. In the quantitative study, we adapted a procedure from Lohse and Peltz [2] to calculate eye movement patterns from eye tracking data. We then analyzed these patterns to determine which patterns are more common amongst programmers reading source code. This analysis led us to various observations about types of eye movement patterns programmers tended to have. We found that these patterns we observed tended to be consistent from method to method for each programmer. We also found that the patterns tended to be shared amongst the entire group of programmers.

II. The Problem

In this paper, we fill a gap in current program literature: there are currently no studies on what *order* programmers read source code for the purpose of summarization. The order of reading is significant to understanding code because it can help reveal which keywords programmers fixate on during summarization tasks. It is currently unknown whether

programmers prefer to read code slowly, taking in each word in order to have an in-depth understanding of the overall work. On the other hand, a rapid scan may be enough to get a decent understanding so that the programmer can move on to the next task. Another unknown is whether programmers like to read code in the order of the components in it or if they prefer to start from the ends of lines and work their way back. Discovering which of these reading patterns are preferred by programmers can have a significant impact on the overall understanding of summarization.

The importance of discovering the order in which code is read can be extended beyond general summarization. In an education setting, if the educator understands how computer science students are reading source code when they are first learning to understand both the specific code and code in general, it is possible that they can better help the students with comprehension [9]. This understanding also has the potential to allow the teacher to better guide the students toward a better path for comprehension if they are reading the code in a pattern that does not fit the general pattern. Finally, if a specific pattern is responsible for maximum comprehension, then educators could also guide students to write code in a way that supports summarization tasks [9].

It is known from related literature that comprehension can be reduced when readers follow a reading pattern that they are used to rather than a pattern better suited for the task [1], [2], [3]. For example, in a study of people reading Yellow Page advertisements, Lohse and Peltz [2] found that the position of advertisements, independent of the quality of the business, had a large impact on choice due to the natural scan from top to bottom and overall reluctance to do exhaustive searching. This means that readers' instincts can have a significant impact on scanning pages for information, even if they know it doesn't completely fulfill their original requirements. By finding the pattern programmers follow when reading source code for the first time, we can also find whether what they are used to doing sometimes dictates some of the areas they look to for information, useful or not, similarly to the Yellow Pages study.

III. Background and Related Work

This section will cover background of eye movement patterns research in general, as well as qualitative methods and reading methods used in the past.

A. How Humans Read

There is comparatively far more literature available on how humans read natural language, than on how we read source code. *Natural language text* is plain text written similarly to a reader's native language [10]. If a reader's native language is English, then a book written in a similar style, such as English, French, Spanish, or German, can be considered natural language text. In contrast, machine languages do not follow the same structure as any human's natural language. While natural languages are written with sentences and paragraphs, machine languages are written as statements and blocks. Natural languages allow for sentences to be written in various ways while still retaining the same meaning, while machine languages force a more precise approach when writing statements.

When humans read natural language text, there are generally two ways it is accomplished: 1) slowly, using return sweeps and thorough reading or 2) rapidly, using skimming and scanning. A *return sweep* is the motion of the eyes moving from the end of one line to the beginning of the next [11], [12]. For it to be a true return sweep, the end of a line should be on one end of a page, while the beginning of the next line is on the opposite side. *Thorough reading* is moving from line to line, making sure to read almost every word in every line [12]. These techniques help ensure that the reader has a complete understanding of the text when reading is complete.

In contrast, skimming and scanning do not ensure a complete understanding. *Skimming* is the act of rapidly reading a section of text, attempting to get a summary of the information [12], [13]. For example, by skimming the introduction of a research paper, one can determine if they want to read the rest of the paper. *Scanning* is the act of rapidly reading a section of text, attempting to gain insight about a particular piece of information [13]. For example, by scanning the surrounding words around an unknown word in prose, a reader may be able to use context clues to discern a sufficient meaning for the unknown word. There are two types of scanning, where the reader finds all instances of a particular repeated word or set of words throughout the text or where someone finds a particular word or set of words and studies the words around them [13]. These techniques help ensure that the reader has a only partial understanding of the text, in the hopes that the partial understanding covers everything important, while saving a reasonable amount of time.

In a few studies, findings have shown humans to read source code different than natural language text [4], [7], [14]. These studies found that programmers reading source code tended to skip around the text looking for specific information and also changed how they read the text depending on their needs or the style of the source code. However, another study found that some source code reading patterns maintain natural language reading traits [15]. This study revealed a tendency to scan through source code in a similar way as readers scan through prose. In this paper, we intend to more directly determine whether programmers' eye movement patterns follow a natural language pattern or not.

B. Eye Movement Patterns

Although there have been several studies of programmers' eye movements, see Section 3.2.3, there are not many studies on patterns of movement. There are also studies in other areas that have conducted experiments for finding the reading patterns of the general public, some of which used eye tracking software to gain more accurate insights. The majority of these studies involve consumers reading about products and/or service. The Norman group[1] recently published a technical

[1]The Norman group is a private training and research company that specializes in user experiences and user interfaces [3].

report describing how people read web pages under various conditions. There have been various reports published with this premise in the past, but this newest issue includes eye tracking evidence to support the claims made about people's reading patterns. This report is primarily intended as a guide for helping web designers improve their site designs, providing information on people's reading patterns while using computers and/or the internet [3]. Other studies have conducted similar experiments, but with physical mediums, such as newspapers and Yellow Pages [1], [2]. These studies also used eye tracking hardware to more accurately follow the eye movements of the participants. These studies mainly focused on the average consumer's attention to companies' ad placement strategies.

1) Yellow Pages Study: The quantitative analysis method we use later in this paper is based on an earlier study of eye movement patterns in advertising, specifically a technique developed for analysing the Yellow Pages[2]. For purposes of background and reproducibility, we describe that paper here. Lohse and Peltz conducted an eye tracking study to see which companies with comparable services would get chosen given certain Yellow Page listings and advertisements [2]. Lohse and Peltz define an eye movement pattern to be the "fixation number sequence", where a fixation is a temporary moment of no eye movement lasting at least 100 milliseconds [2]. The experiment was run in 60 minute sections, with only one participant per section. The participant sat in a room with a researcher, but calibrated and used the eye tracker themselves. The participant then read the instructions on what to look for and how to switch between pages, if needed. There was no extra special equipment, i.e. chin stabilizers, neck braces, etc., used during the experiment apart from the eye-tracking device [2]. The results from the study revealed that the businesses that were lower on the list alphabetically or had smaller ads were not chosen to complete the service simply because the participants would finish their search long before reaching these companies. After analyzing the reading patterns, Lohse and Peltz found that people did have a systematic way of looking for a business to use, which was mainly using brute force by searching alphabetically or in order from largest to smallest ads [2]. They also found that some participants did not even fully read the company names or advertisements as they scanned them — if they didn't see what they wanted right away, they quickly moved on [2].

2) Eye-Tracking in Program Comprehension: Various eye tracking studies have been conducted in computer science [3], [6], [7], [14], [15]. One early study was conducted by Crosby *et al.*. This study determined that there is a difference between reading source code and reading natural language text [4]. Programmers tended to fixate on important keywords and sections rather than thoroughly reading the full length of text [4]. In two separate studies, Bednarik *et al.* found that experienced programmers tended to fixate on function and expressions outputs, while inexperienced programmers tended to repetitively fixate on the same sections [7], [14]. Uwano *et al.* found that programmers discovered bugs more easily when they took more time to scan the source code before fixating on a particular section [15]. Sharif *et al.* confirmed that scan time and bug detection time are correlated [6].

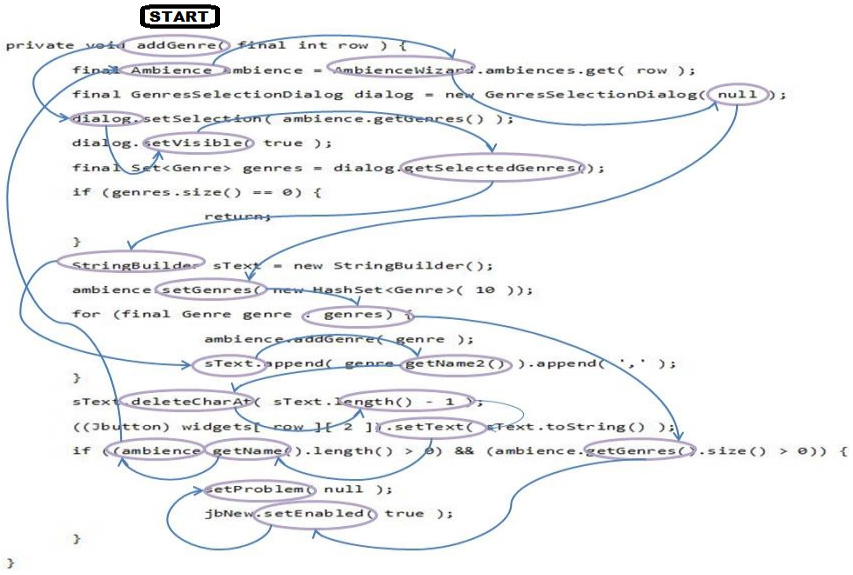
3) Our Previous Eye-Tracking Study: We conducted a previous eye tracking study in which we examined what keywords in source code programmers honed in on as they scanned a method for comprehension. Keywords were also separated into categories (signature, control flow, and invocation) in order to determine if a specific section of the method was considered more helpful for overall comprehension [8]. The methodology of the study was to have the participants spend limited time reading various methods and writing short, descriptive summaries about each one. Specifically, we had each participant sit in a room alone for 60 minutes. The participants themselves were all professional programmers with a range from 6 to 27 years of experience. The methods they read were all written in Java and were around 22 LOC each. The computer the participants used had an eye-tracking system built into the monitor that capture at 120 Hz. Each participant was required to calibrate the eye-tracker before beginning the exercise. The calibration helped the eye-tracker understand the participant's normal head and eye positioning. Then the participant was given an example method that guided them through the process needed to complete the exercise. There was no additional equipment used or human interaction attempted during the 60 minute section [8]. The results of the study showed that programmers tend to favor the signature over the body of the method, and they tend to favor words not associated with the control flow [8]. It is the eye-tracking data from this study that we will use as input for finding source code reading patterns.

C. Program Comprehension

There are many studies on program comprehension. Holmes *et al.* and Ko *et al.* have found that many of these studies focus on the strategies and techniques used by programmers based on the information they want [16], [17]. For example, some programmers follow an "opportunistic" strategy aimed at finding only the section of code that is needed during maintenance tasks [18], [19], [20], [21]. In contrast, a "systemic" strategy aims at understanding how different sections of source code interact with each other [22], [23]. These strategies are useful to understand, but they are specific to certain program comprehension tasks. Unfortunately, these studies have not looked at the strategies programmers use when reading source code in general. Mayer *et al.* speaks about some programmers following "bottom-up comprehension", meaning that they prefer to read source code to create mental abstractions to understand the system [24]. However, they still do not give an explanation of how programmers specifically read the source code in order to accomplish this goal. The findings presented in this paper aim to further explain this bottom-up comprehension, especially since this form of comprehension can be directly affected by programmers eye movement patterns.

[2]The Yellow Pages is a telephone directory of businesses, usually published annually, and distributed for free to all residences and businesses within a given coverage area.

Fig. 1: A sample java method after completing qualitative analysis. The circled keywords were ones fixated on for 300ms or longer. The bold arrows represent a transition in focus from one keyword to another. (In this sample, there are approximately 25 of the beginning transitions.) This sample begins with a fixation on the append function call within the for-loop and ends length function call just after the for-loop. There are two important aspects of this sample to notice that are shared amongst many of the patterns. First, notice that many of the transitions are within two lines of the originating keyword (i.e., sectional scanning). This is a common occurrence during the beginning of a method reading. Second, notice that many keywords on the same line are fixated on, but that there are not many return sweeps. This is a sign that the programmer is skimming the source code. Lastly, notice that there does not appear to be a regular pattern of the direction of reading.



IV. Qualitative Study Design

In this section, we describe our research questions, the methodology of our qualitative study, and details of the settings for the study.

A. Research Questions

The goal of this qualitative study is to obtain evidence of eye movement patterns that can be tested later in the quantitative study. Towards this goal, we propose four Research Questions (RQ):

- RQ_1 Do programmers tend to read code from top to bottom and from left to right, similar to English text?
- RQ_2 Do programmers tend to skim or thoroughly read the code?
- RQ_3 Do programmers tend to jump sections rapidly or do they finish a specific section before moving to the next ones?
- RQ_4 Do programmers tend to read code similar to other programmers?

The rationale for the first RQ is that a reasonable way for programmers to read code might be the same way they read natural language text. Answering RQ_1 will contribute to the debate on whether code comprehension is similar to text comprehension. At present, different studies have come to different conclusions [4], [25]. The rationale behind RQ_2 is that evidence of programmers skimming through code has been found before, and we want to see if this finding is corroborated by the eye tracking data [5]. Whereas RQ_2 tests skimming over keywords, RQ_3 tests whether programmers skim over whole sections. The other option is that programmers could prefer finding a main keyword, reading around it for context clues, and then moving on to the next main keyword once they fully understand the first. RQ_4 is to see if the findings from each individual programmer is similar enough to each other to suggest that the eye movement patterns we found are generalizable.

B. Methodology

Our methodology was to manually inspect the eye movement patterns by drawing the patterns on paper. This strategy enabled us to rapidly identify the patterns for the purposes of this qualitative study. We use data from a previous eye-tracking study (see Section III-B3). We hand-draw a circle around the first word read of a method and then move to the second word and circle that keyword. From there, we connect the two keywords with a line and arrow, indicating the order (see Figure 1 for an example). We continue doing this for every word fixated on in each method. We repeated this process for 5 participants and 4 methods per participant. These participants and methods were randomly selected from our original results pool. After all of the methods are completed, the lines with arrows are analyzed to distinguish any clear patterns.

C. Threats to Validity

Like all studies, this qualitative study has some threats to validity. One threat is that methods were printed out on standard printer paper in order to draw the connections. It is possible that the small size of the paper coupled with the large amount of keyword fixations throughout the eye-tracking study could have caused some arrowed lines to cross in ways that made them confusing. We cannot rule out the possibility that some words may have been seen in a slightly different order than originally recorded. However, this threat was minimized due to the shear number of words, making a few mistakes acceptable.

Another source of a threat is that a human researcher was responsible for drawing the circles and lines. It is possible that the researcher made some mistakes due to fatigue or stress. We minimized this threat by requiring incremental sessions of drawing and analysis to reduce a stress-causing workload.

V. Qualitative Study Results

In this section, we present our results to each research question. We also present some examples for how we formulated our answers and our rationale behind those answers.

A. RQ_1: Source Code and Natural Language Text Comparison

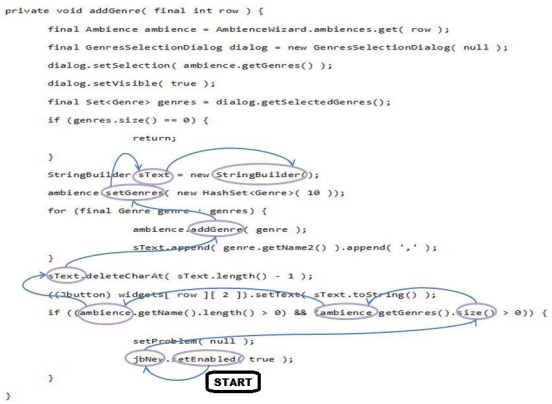
From our analysis, we found that the programmers read source code similarly to their native language (in this case, English). All programmers in our study tended to read code from left to right, e.g., performed return sweeps. However, the programmers did not always read from top to bottom. They also read from bottom to top. Starting at the top or the bottom appears to be a split preference of the programmer. Among our participants, both of the preferences occurred with about the same frequency.

Figure 2(a) and Figure 2(b) are examples of both source code reading patterns described. The bold arrow lines in Figure 2(a) show a small section of code reading that closely resembles natural language text reading. The bold arrow lines in Figure 2(b) show a small section of code reading that is similar to natural language text reading, except that it runs from bottom to top.

Our rationale is that programmers tend to read from left to right because the majority of context clues in source code are relatively further to the left. Even if the important keywords are all on the right side of the method, they may not provide enough information on their own. For example, consider an important keyword in Figure 2(a) is getGenres. On line 4, getGenres is passed as a parameter to setSelection, which is being invoked by dialog. Just seeing that getGenres is being passed as a parameter is not enough to understand its purpose. Discovering why setSelection needs getGenres and why dialog would call setSelection is necessary for understanding the importance of getGenres.

Another possibility is that programmers use the pattern of top to bottom reading and the alternative pattern of bottom to top reading with the same intentions, but for different reasons.

(a) An example of reading from left-to-right and top-to-bottom.



(b) An example of reading from left-to-right and bottom-to-top.

Fig. 2: Examples of the two reading styles seen in the qualitative exploration. The exploration found programmers to be almost evenly split between these two styles, with a slight tendency towards the top-to-bottom approach. It is important to note that these are simplified versions of these two styles. The exploration revealed that these styles are often mixed within a single method reading. Also, the return sweeps shown here are mostly single line movements, while a return sweep can include multiple skipped lines.

The intention of both patterns is to help make the overall purpose of the method more clear, more quickly. Reading from top to bottom ensures that one of the first areas read is the method signature, which contains the method name, the return type, and the input parameters. All of this information within the signature provides a significant amount of clues for determining the overall purpose. However, the end of the method also tends to provide significant clues to the overall purpose. As can be seen in Figure 2(b), it is sometimes possible for the end of a method to provide more important information than the signature.

B. RQ_2: Reading Technique: Skimming versus Thorough

During our analysis, we discovered that programmers tend to skim source code as opposed to thoroughly reading through it. As we mapped the sequence of keyword views, we kept track of how quickly the programmers moved from word to word. If the participant maintained focus on a particular word for over 1000ms, we considered them to be inspecting it. It was common for some keywords to cause programmers to hold their focus from 1000ms to upwards of 6000ms. However, it was much more common for programmers to leave and return to these important keywords several times during their reading process, rather than hold their gaze on any particular keyword for an extended period of time.

Our interpretation is that programmers skim through code instead of read through it because of time pressures. Reading is certainly a more thorough way of both determining the purpose of this keyword and ensuring that a particular keyword is important to the method. However, skimming may help prevent reading of unnecessary words, with the condition that some keywords may need to be revisited multiple times to determine their purpose and level of importance.

C. RQ_3: Scanning Technique: Disorderly versus Sectionally

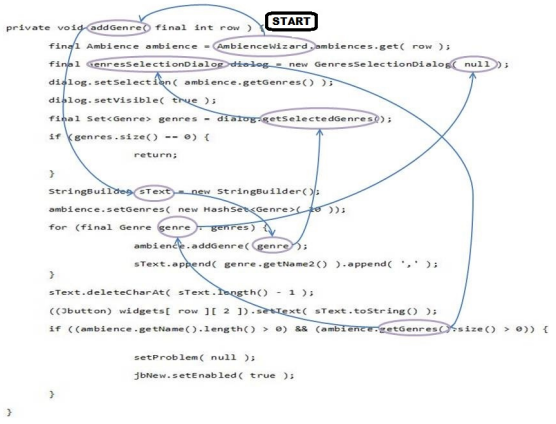
From our analysis, we found that programmers prefer to jump among important keywords (e.g., to scan disorderly), rather than thoroughly inspect the sections surrounding the keywords (e.g., to scan sectionally). However, a significant portion of the participants began scanning some of their methods using the sectional technique before eventually using the disorderly technique. The sectional technique tended to only last until the programmer had scanned the entire method with this technique exactly one time.

Figure 3(a) and Figure 3(b) are examples of both of these techniques. The bold arrow lines in Figure 3(a) show a disorderly scan through the method. The bold arrow lines in Figure 3(b) show a sectional scan through a method.

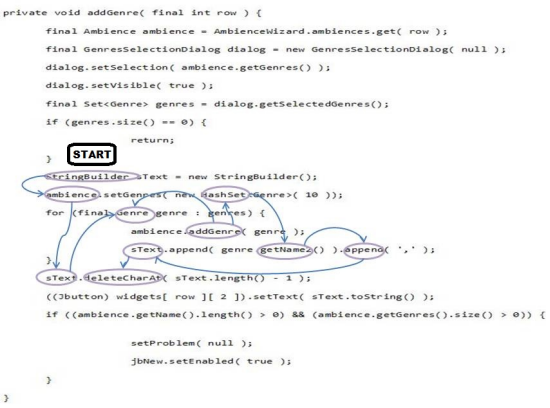
We believe the reason that programmers prefer to jump among important keywords is similar to the reason that they prefer to skim code. Reading entire sections at a time may lead to a better overall understanding, but may also waste time through the processing of unimportant words just used for connecting important keywords together. Programmers using the sectional technique during the first pass of a method may likely be a way of gauging the importance of specific keywords to increase the efficiency of the disorderly technique. In a sense, the first sectional scan gives some order to the disorderly technique.

D. RQ_4: Similarities: Programmer by Programmer

We compared the visible patterns of each programmer and determined that the patterns were similar, but not entirely the same. As stated for RQ_1, the preferences in reading direction appeared to be split almost in half. As also seen for both RQ_2 and RQ_3, most of the programmers tended to use the faster technique. However, one programmer did tend to read more thoroughly and hold his gaze for longer on certain keywords for all of his methods, and a separate programmer tended to read in a more sectional manner than his fellow participants.



(a) An example of scanning disorderly.



(b) An example of scanning sectionally.

Fig. 3: Examples of the two scanning techniques. The qualitative exploration found that most programmers prefer the disorderly approach. However, some programmers used the sectional approach near the beginning of a method reading. The exploration also found that, although uncommon, a few programmers read a couple of methods using mostly sectional scanning.

Although our analysis shows that a single eye movement pattern may not fit all programmers, we interpret these findings as a good reason for investigating these patterns further. These findings show that, although all programmers may not have the exact same code reading patterns, there may be a significant amount of them with comparable patterns. We also believe that since this qualitative analysis was done on a subset of methods, further study is warranted.

E. Summary of Qualitative Results

We derive three main interpretations from our qualitative analysis results. The first interpretation is that programmers read through source code roughly the same way they read through natural language text. They perform return sweeps, just as they would with text. However, the difference is that programmers have the liberty of starting at the top or the bottom depending on which they believe will work best for overall comprehension. Since the end of a method can sometimes contain more important information than the method signature, it is entirely possible that different methods may be better understood using one pattern over the other. The second interpretation is that programmers prefer to be thorough as little as possible. For both the reading and scanning techniques, the less thorough of the techniques was preferable. The thorough techniques are more time-consuming, but also more definitive. The third major interpretation is that the eye movement patterns of programmers are similar compared to one another, meaning that our findings here are more likely to be generalizable. Our qualitative conclusion is that programmers' main reading pattern is skimming and jumping from left to right starting at either the top or the bottom, depending on preference and method.

VI. Quantitative Study Design

In this section, we describe our research questions, the methodology of our quantitative study, and details of the settings for the study.

A. Research Questions

The goal of this quantitative study is to further explore the questions proposed in the qualitative study. Towards this goal, we have proposed four Research Questions (RQ):

- RQ_5 To what degree do programmers prefer to read source code similar to English text?
- RQ_6 To what degree do programmers prefer to skim source code?
- RQ_7 To what degree do programmers prefer to jump between sections in source code rapidly?
- RQ_8 To what degree are programmers' reading patterns similar to other programmers' reading patterns?

The rationale behind RQ_5, RQ_6, RQ_7, and RQ_8 is based on what we found in the qualitative section. We found that programmers do not prefer one reading technique completely over another. These partial preferences require us to determine the extent of each preference. Specifically, for RQ_5, we need to determine if programmers' patterns are split between reading code like English text or reading it the opposite. Even if the split is not even, as was seen in the qualitative study, the split may show RQ_5 to be insignificant for determining eye movement patterns. For RQ_6, we need to determine if programmers do prefer to skim through keywords. For RQ_7, we need to determine if programmers also prefer skimming through sections of code as they seem to through keywords. The findings of the qualitative study suggest that programmers tend to skim most of the time in both cases. Finding the answer to RQ_8 is especially important. In the qualitative study, we saw that the majority of reading patterns agreed, but some people appeared to read the code differently for specific methods. This appearance of separate reading patterns for some programmers means that the similarities between eye movement patterns may not be statistically significant. If this is the case, then programmer eye movement patterns may not be as helpful for understanding summarization task completion. Since the qualitative study was run on a subset of the overall data, there is the possibility that the patterns seen were coincidental.

B. Methodology

The methodology of our study is to extract the order of eye fixations of different programmers, and then compute the similarity of those orders of fixations. We adapted the technique advocated by Lohse and Peltz [2] in their study of eye fixation patterns in advertising (see Section III-B1). Specifically, for each research question, we:

1) For RQ_5, created a mapping to represent paths such as "left to right" and "bottom to top" as values. Using this mapping, we can determine the overall percentage that programmers use a natural language text reading pattern.
2) For RQ_6, directly used the time values during and between keyword gazes. With these time values, we can determine the percentage that programmers stay focused on certain keywords and slowly read through them.
3) For RQ_7, created a mapping to represent paths such as "section 1 to section 3" and "section 5 to section 2" as values. Using this mapping, we can determine the overall percentage that programmers tend to stay in each section as long as possible versus constantly switching between sections.
4) For RQ_8, used a combination of the values we created for RQ_5 through RQ_7 for each individual's methods to create overall pattern values.

As Lohse and Peltz [2] point out, this procedure combines individual programmer eye movements to inform us about patterns of eye movement in general.

C. Statistical Tests

We compared direction, skimming, and transfer patterns using the Wilcoxon signed-rank test [26]. This test is non parametric and paired, and does not assume a normal distribution. It is suitable for our study because we compare patterns paired for each method and because our data may not be normally distributed. We determine similarities in patterns between individuals by using one-way ANOVA. It is suitable for our study because all methods used were similar in size and type, but are not necessarily the same.

D. Threats to Validity

This quantitative study has three major threats to validity. The first threat is that there is not a large amount of data, so we cannot ensure that our data falls on a normal distribution and is fully generalizable. The second threat is that the programmers participating in the eye-tracking study may have altered their normal reading patterns due to the stress and circumstances

TABLE I: Statistical summary of the results for RQ_4, RQ_5, and RQ_6. Wilcoxon test values are U, U_{expt}, and U_{vari}. Decision criteria are Z, Z_{crit}, and p. A "Sample" is one programmer for one method.

RQ	H	Pattern Type	Samples	$\tilde{x}$	Vari.	U	U_{expt}	U_{vari}	Z	Z_{crit}	p
RQ_5	H_1	Natural Unnatural	132 132	1.001 0.999	0.000196 0.000196	4199	4064	172720.000	1.645	0.325	0.373
RQ_6	H_2	Skim Thorough	132 132	0.979 0.021	0.001600 0.001600	8778	4389	192514.875	1.645	>3.75	<1e-3
RQ_7	H_3	Disorderly Sectionally	132 132	0.698 0.302	0.002116 0.002116	8771	4389	193847.375	1.645	>3.75	<1e-3

that naturally appear during a research study. The third threat is that the programmers may have had their eyes partially focused on one word while also looking at another. This is a phenomenon Rayner refers to as the peripheral span [12].

VII. Quantitative Study Results

In this section, we present our results to each research question. We also present some examples for how we formulated our answers and our rationale behind those answers.

A. RQ_5: Source Code and Natural Language Text Comparison

We found no statistical evidence that programmers tend to read using their English text reading patterns. In fact, we found that programmers read from left-to-right or from top-to-bottom almost exactly as much as they read from right-to-left or from bottom-to-top. On average, programmers followed a natural language text reading pattern about 49% of the time. To determine significance, we computed the percentage that programmers tended to read left-to-right or top-to-bottom versus the percentage that programmers tended to read right-to-left or bottom-to-top. We then posed H_1:

H_n The difference between the computed metric for reading code the same as reading English text and reading code differently from reading English text is not statistically-significant.

Using the Wilcoxon signed-rank test, we could not reject the null hypothesis (see table 1). These results indicate that programmers do not necessarily prefer reading code using a similar pattern to how they read natural language text.

B. RQ_6: Reading Technique: Skimming versus Thorough

We found statistically significant evidence that programmers tend to skim code rather than read it in-depth. On average, programmers thoroughly read about 10% of each method. To determine significance, we computed the percentage that programmers tended to thoroughly read the code versus the percentage that programmers quickly skimmed through it. We then posed H_2:

H_n The difference between the computed metric for thoroughly reading code and skimming code is not statistically-significant.

Using the Wilcoxon signed-rank test, we rejected the null hypothesis (see table 1). These results indicate that programmers do not thoroughly read through source as much as they quickly skim through it.

C. RQ_7: Scanning Technique: Disorderly versus Sectionally

We found statistically significant evidence that programmers tend to jump around code in a seemingly disorderly manner when reading code, as opposed to moving from section to section as they read. On average, programmers read in sections about 25% of the time. To determine significance, we computed the percentage that programmers stayed in a section when reading code versus the percentage that programmers jumped from section to section. We then posed H_3:

H_n The difference between the computed metric for reading code in a sectional manner and reading code in a disorderly manner is not statistically-significant.

Using the Wilcoxon signed-rank test, we rejected the null hypothesis (see table 1). These results indicate that programmers prefer to read code by jumping between sections rather than reading code one section at a time.

D. RQ_8: Similarities: Programmer by Programmer

We found statistically significant evidence showing that all of the programmers followed very similar patterns in reading overall. When comparing the results of the three research questions above between each programmer, the results were almost identical. To determine significance, we computed a representative value for a reading pattern for every programmer and every method. Then, we compared each value to see how similar they were to each other. We then posed H_4:

H_n The similarities amongst the computed metric for a programmer's reading pattern for all programmers and methods are not statistically-significant.

Using one-way ANOVA test, we rejected the null hypothesis (see table 2). These results indicate that programmers do all read code in a similar manner as each other.

E. Summary of Quantitative Results

We derive three main interpretations from our quantitative analysis results. The first interpretation is that programmers read through source code from all directions. As indicated by the statistical results, programmers appear to have no

TABLE II: Statistical summary of the results for RQ_7. ANOVA test values are Degrees of Freedom (DF), Sum of Squares (SS), Mean Sum of Squares (MS). Decision criteria are F, F_{crit}, and p. A "Sample" is one programmer for one method.

RQ	H	Pattern Type	Samples	$\tilde{x}$	Vari.	Source	DF	SS	MS	F	F_{crit}	p
RQ_8	H_4	All	132	1.323	0.0025	Model	8	0.160	0.020	14.168	2.009	<1e-3
						Error	123	0.174	0.001			
						Corrected	131	0.333				

preference for when they start and stop reading a section of code. This lack of preference may be an artifact of both lines of source code not have as formal of a structure as lines of natural language text and, for the purposes of summarizing code, some of the more important keywords may be towards the right and bottom of the overall code. The second interpretation is that programmers prefer to be thorough as little as possible. For both the reading and scanning techniques, the less thorough of the techniques was preferable. This preference of a less thorough behavior may be from the fact that the more thorough techniques are more time-consuming. The third interpretation is that, in general, programmers follow similar reading patterns compared to other programmers.

VIII. Discussion

Our paper contributes to the software engineering literature with empirical evidence of programmer behaviour during summarization tasks. These findings show that programmers follow reading patterns that tend to give them a quicker understanding of the program, as opposed to a more in-depth understanding. First, we found that programmers tend to read with a natural text progression, from left-to-right or from top-to-bottom, almost exactly half of the time that they are reading source code. This finding shows that programmers don't follow the natural language reading pattern exactly, but they do not completely ignore it either. Second, we found that programmers tend to read by skimming the code rather than slowly, thoroughly going through it. Thirdly, we found that programmers tend to read by jumping around to different important keywords spread across the code rather than finishing one section of code at a time before moving on to the next.

One potential explanation that programmers don't necessarily follow natural language text reading patterns is because of the lack of bias towards reading code in any particular way. It is possible that, when it comes to natural language text, most people are educated from an early age to read it in a very specific way in order to comprehend it best. People are taught to read it from the beginning to the end, thoroughly, in order to best determine the meaning of words, phrases, and the work as a whole. People are also taught to write from beginning to end, getting all of our thoughts on paper in a structured, linear way. However, when it comes to source code, people are likely not taught to look at it a certain way when attempting comprehension. Our analysis could be used to improve the overall educational process for programming.

An additional benefit of this work is the future improvement of source code summarization tools. Different source code summarization tools have generated natural language summaries [27], [28], [29], [30], [31]. These approaches have largely been built from program comprehension studies of tasks other than summarization or studies that did not take reading patterns into mind. Where there were no studies that provided information, these tools made assumptions about how programmers read code and what programmers need in summaries. We have shown that programmers do follow certain reading patterns during summarization tasks. Our work can assist code summarization research by providing a guide to what order programmers need to see keywords. At the same time, our work may assist in creating metrics for source code comment quality [32] by providing evidence about which keywords the comments should describe and where in the code these comments should be located.

Although this work does show important evidence that contributes to software engineering, there is some future work to expand upon this research that we believe will be beneficial. A major area of future work is to create an algorithm that can accurately predict the order in which a programmer will read keywords in a given source code method. If the algorithm could at least predict the flow of a programmer's summarization attempt, it could go a long way to helping determine what keywords should be highlighted or included in a comment or summary. This algorithm could incorporate our findings from this study.

Another area of future work is to improve upon this study by considering some of the study's limitations. The results were relatively stable for how similar each programmer's reading patterns were compared to each other; however, it would make the results more concrete if we were able to incorporate more programmers and a wider range of source code methods. Also, due to the software used to run the eye-tracking machine, the source code methods were limited to 22 LOC or less, restricting the types of methods that could be used. Finally, we would like to also expand the qualitative work by using a heat map program or eye movement classification software to add an automated approach to our manual one. These improvements would greatly enhance our results.

IX. Conclusion

We have presented a qualitative and quantitative study about programmers during source code summarization. We have explored eight Research Questions aimed at understanding the patterns programmers follow when reading and attempting to summarize code. We showed that programmers tend to read source code a little differently than they read natural language text. We showed programmers prefer to read source

code using rapid techniques such as skimming and jumping around, as opposed to more thorough techniques, such as in-depth and sectional reading. Our findings lead us to conclude that programmers not only follow specific patterns when they read and summarize source code, but they also all tend to use similar patterns compared to each other.

ACKNOWLEDGMENTS

We thank and acknowledge the 10 Research Programmers at the Center for Research Computing at the University of Notre Dame for participating in our original eye-tracking study. Any opinions, findings and conclusions, or recommendations expressed in this paper are those of the authors.

REFERENCES

[1] J. Holsanova and K. Holmqvist, "Entry points and reading paths on newspaper spreads: comparing a semiotic analysis with eye-tracking measures," 2006.

[2] G. Lohse and N. Peltz, "Consumer eye movement patterns on yellow pages advertising," 2009.

[3] N. N. Group, "How people read on the web: The eyetracking evidence," Nielson Norman Group, Tech. Rep., 2013.

[4] M. E. Crosby and J. Stelovsky, "How do we read algorithms? a case study," *Computer*, vol. 23, no. 1, pp. 24–35, Jan. [Online]. Available: http://dl.acm.org/citation.cfm?id=77577.77580

[5] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *Proceedings of the 2012 International Conference on Software Engineering*, ser. ICSE 2012, pp. 255–265. [Online]. Available: http://dl.acm.org/citation.cfm?id=2337223.2337254

[6] B. Sharif, M. Falcone, and J. I. Maletic, "An eye-tracking study on the role of scan time in finding source code defects," in *Proceedings of the Symposium on Eye Tracking Research and Applications*, ser. ETRA '12. New York, NY, USA: ACM, 2012, pp. 381–384. [Online]. Available: http://doi.acm.org/10.1145/2168556.2168642

[7] R. Bednarik and M. Tukiainen, "An eye-tracking methodology for characterizing program comprehension processes," in *Proceedings of the 2006 symposium on Eye tracking research & applications*, ser. ETRA '06. New York, NY, USA: ACM, 2006, pp. 125–132. [Online]. Available: http://doi.acm.org/10.1145/1117309.1117356

[8] P. Rodeghero, C. McMillan, P. W. McBurney, N. Bosch, and S. D'Mello, "Improving automated source code summarization via an eye-tracking study of programmers," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE '13, 2013.

[9] N. Bosch, Y. Chen, and S. D'Mello, "It's written on your face: Detecting affective states from facial expressions while learning computer programming," in *Intelligent Tutoring Systems*, ser. Lecture Notes in Computer Science, S. Trausan-Matu, K. E. Boyer, M. Crosby, and K. Panourgia, Eds. Springer International Publishing, 2014, vol. 8474, pp. 39–44. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-07221-0_5

[10] D. R. Swanson, "Searching natural language text by computer," 1960.

[11] J. Hofmeister, D. Heller, and R. Radach, "The return sweep in reading," in *Current Oculomotor Research*, W. Becker, H. Deubel, and T. Mergner, Eds. Springer US, 1999, pp. 349–357. [Online]. Available: http://dx.doi.org/10.1007/978-1-4757-3054-8_49

[12] K. Rayner and A. Pollatsek, *The Phsycology of Reading Abingdon*, 2013.

[13] F. Grellet, *Developing Reading Skills*, twenty-ninth printing ed. Cambridge, UK: Cambridge University Press, 2010.

[14] R. Bednarik and M. Tukiainen, "Temporal eye-tracking data: evolution of debugging strategies with multiple representations," in *Proceedings of the 2008 symposium on Eye tracking research & applications*, ser. ETRA '08, New York, NY, USA, 2008, pp. 99–102. [Online]. Available: http://doi.acm.org/10.1145/1344471.1344497

[15] H. Uwano, M. Nakamura, A. Monden, and K.-i. Matsumoto, "Analyzing individual performance of source code review using reviewers' eye movement," in *Proceedings of the 2006 symposium on Eye tracking research & applications*, ser. ETRA '06. New York, NY, USA: ACM, 2006, pp. 133–140. [Online]. Available: http://doi.acm.org/10.1145/1117309.1117357

[16] R. Holmes and R. J. Walker, "Systematizing pragmatic software reuse," *ACM Trans. Softw. Eng. Methodol.*, vol. 21, no. 4, pp. 20:1–20:44, Feb. 2013. [Online]. Available: http://doi.acm.org/10.1145/2377656.2377657

[17] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Trans. Softw. Eng.*, vol. 32, no. 12, pp. 971–987, Dec. 2006. [Online]. Available: http://dx.doi.org/10.1109/TSE.2006.116

[18] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer, "Example-centric programming: integrating web search into the development environment," in *Proceedings of the 28th international conference on Human factors in computing systems*, ser. CHI '10. New York, NY, USA: ACM, 2010, pp. 513–522. [Online]. Available: http://doi.acm.org/10.1145/1753326.1753402

[19] J. W. Davison, D. M. Mancl, and W. F. Opdyke, "Understanding and addressing the essential costs of evolving systems," *Bell Labs Technical Journal*, pp. 44–54, 2000.

[20] G. Kotonya, S. Lock, and J. Mariani, "Opportunistic reuse: Lessons from scrapheap software development," in *Proceedings of the 11th International Symposium on Component-Based Software Engineering*, ser. CBSE '08, Berlin, Heidelberg, 2008, pp. 302–309. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-87891-9_20

[21] A. Lakhotia, "Understanding someone else's code: analysis of experiences," *J. Syst. Softw.*, vol. 23, no. 3, pp. 269–275. [Online]. Available: http://dx.doi.org/10.1016/0164-1212(93)90101-3

[22] A. J. Ko and B. A. Myers, "A framework and methodology for studying the causes of software errors in programming systems," *Journal of Visual Languages and Computing*, vol. 16, no. 12, pp. 41 – 84, 2005. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1045926X04000394

[23] D. C. Littman, J. Pinto, S. Letovsky, and E. Soloway, "Mental models and software maintenance," *Journal of Systems and Software*, vol. 7, no. 4, pp. 341 – 355, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0164121287900331

[24] B. Shneiderman and R. Mayer, "Syntactic/semantic interactions in programmer behavior: A model and experimental results," *International Journal of Computer & Information Sciences*, vol. 8, no. 3, pp. 219–238, 1979. [Online]. Available: http://dx.doi.org/10.1007/BF00977789

[25] J. Siegmund, C. Kästner, S. Apel, C. Parnin, A. Bethmann, T. Leich, G. Saake, and A. Brechmann, "Understanding understanding source code with functional magnetic resonance imaging," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 378–389. [Online]. Available: http://doi.acm.org/10.1145/2568225.2568252

[26] D. A. Wolfe and M. Hollander, "Nonparametric statistical methods," *Nonparametric statistical methods*.

[27] H. Burden and R. Heldal, "Natural language generation from class diagrams," in *Proceedings of the 8th International Workshop on Model-Driven Engineering, Verification and Validation*, ser. MoDeVVa. New York, NY, USA: ACM, 2011, pp. 8:1–8:8. [Online]. Available: http://doi.acm.org/10.1145/2095654.2095665

[28] S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: approach for unsupervised bug report summarization," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, ser. FSE '12. New York, NY, USA: ACM, 2012, pp. 11:1–11:11. [Online]. Available: http://doi.acm.org/10.1145/2393596.2393607

[29] L. Moreno, J. Aponte, S. Giriprasad, A. Marcus, L. Pollock, and K. Vijay-Shanker, "Automatic generation of natural language summaries for java classes," in *Proceedings of the 21st International Conference on Program Comprehension*, ser. ICPC '13.

[30] G. Sridhara, "Automatic Generation of Descriptive Summary Comments for Methods in Object-oriented Programs," Ph.D. dissertation, University of Delaware, Jan.

[31] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Generating parameter comments and integrating with method summaries," in *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ser. ICPC '11. Washington, DC, USA: IEEE Computer Society, 2011, pp. 71–80. [Online]. Available: http://dx.doi.org/10.1109/ICPC.2011.28

[32] D. Steidl, B. Hummel, and E. Juergens, "Quality analysis of source code comments," in *Proceedings of the 21st International Conference on Program Comprehension*, ser. ICPC '13, 2013.