Do Experienced Programmers put too Much Confidence in Comments?

Elia Eiroa-Lledo, Abby Bechtel, Emily Daskas, Lily Foster,
Raha Pirzadeh, Katie Rodeghiero, Erik Linstead

Fowler School of Engineering, Chapman University
E-mail: mlatlab@chapman.edu

Abstract

We present the results of a small eye-tracking study of novice and experienced programmers asked to deduce the output of Python and Java code snippets. We observe experienced programmers paying more visual attention to code comments, and when the comments provided are purposefully misleading, the experienced programmers are more likely to incorrectly describe the code output than their novice counterparts. While preliminary, these results suggest that experienced programmers, who are well-trained in the importance of documentation as part of the software development process, may have an initial tendency to put too much confidence in code comments when faced with program comprehension tasks.

Keywords- program comprehension, eye tracking, code comments

1. Introduction

As part of their formal training, software engineers learn that well-commented, easily comprehensible code is equally as important as code that runs efficiently and correctly. This is reinforced in professional practice, where software deliverables are accompanied by substantial amounts of documentation, both inside the source code as comments, and outside the source code as API specifications and reference manuals. Such documentation provides valuable insight into the function and usage of code, and becomes a critical resource for programmers who must thoroughly understand pieces of existing code as part of software development activities. This leads to two natural questions.

- How much attention do programmers pay to source code comments when tasked with comprehending a piece of code they did not write?

DOI reference number: 10.18293/SEKE2020-063.

- Does experience level impact the level of reliance or confidence a programmer has in comments when faced with a program comprehension task?

We conducted a small sample (N=14) eye-tracking study to gain further insight into how developers read Python and Java code and, in particular, what captures their visual attention as part of program comprehension tasks. We observed that when experienced programmers are presented with code and asked to determine its output, they often focus on comments before moving on to code to formulate their response. In contrast, novice programmers often focus on the source code itself. When subsequently asked to verbally describe the purpose of a code snippet they had just read, experienced programmers in our study were more likely to base their answer on the comments accompanying the code, even if the comments were wrong! In this paper, we present some visual attention patterns of programmers of different skill levels from our study and explore whether experience with programming and comfort reading code can lead to too much faith in source code comments. Our results, while preliminary, indicate that, on average, experienced programmers spend a significantly longer amount of time focusing on documentation during a program comprehension task relative to novices and are more likely to be initially mislead by comments that are technically incorrect.

2. Experimental Design and Data Collection

We recruited a convenience sample of 14 programmers from a computer science department at a University. Hence, all of our participants have had formal training in programming and software development at the collegiate level. We segmented our participants into two groups: novice and experienced programmers. We defined a novice coder as someone who has been coding for less than four years and has not had industry experience as a software engineer. We defined an experienced coder as someone who has had more than four continuous years of programming usage and has had industry experience as a software engineer. Our sample

was evenly split between novice and experienced programmers but, there was only one female programmer in each of the categories.

All of the participants were comfortable with coding in Java and Python. Our experts averaged 29 years in age with an average coding experience of 12.3 years (median = 7 years) and reported spending around 10 hours of their free time coding per week. The novice coders averaged 21 years in age with an average coding experience of 1.9 years (median = 1 year) and reported spending around 2 hours of their free time coding per week.

To begin, each participant of the study took a survey which included questions pertaining to their demographics, educational background, and programming experience. After answering the survey, each participant was set up at a computer monitor where the test would take place and their eyes were calibrated on the EyeLink 1000 Plus eye-tracker. After this, a small set of instructions was read to the participant and the test began. There were four questions of interest in our study, all of which contained a block of code with an in-line comment. Three of the four questions had comments which were intentionally misleading. Other types of coding questions were randomly dispersed throughout the test as well. Before viewing each block of code, the participants read instructions which prepared them for what they were about to see and how they were expected to answer. The participants moved at their own pace, clicked through the questions, and they answered verbally when they were ready to move on. Their responses were recorded via a microphone. This allowed them to read the code at their own speed and with their natural eye-gaze patterns, without any time pressure or the need to write anything down.

The EyeLink 1000 Plus is a fast, high-precision, video-based eye-tracking device which follows a person's pupils and tracks their eye-gaze. This data is then available for data visualization via the EyeLink DataViewer. This data allowed us to gather information beyond the verbal answers given to us by the participants. We were able to observe, for instance, if they ignored the comments and only looked at the code or if they read both but, based on their verbal answers, decided to put their trust in one over the other. Ultimately, the data we collected was a compilation of the eye-tracker data and the recorded responses from the participants.

3. Analysis and Discussion

We asked our participants to describe the output of some code snippets; what they did not know is that some of the code had misleading comments. When inspecting the code, novice coders tended to read both the code and the comments before answering the prompt. Although some based their answer on the misleading comments at first, most of them either quickly backtracked their answer correcting it or, after the first question, caught on to the fact that the comments were not correctly describing the function of the code. Some novice coders did fall for the comments every time. This, we deduce, is because they either were not making a concerted effort to correctly answer the questions, or that they were unable to deduce the output of the code regardless of the comments.

The experienced coders, on the other hand, were extremely confident in their answers and moved on as soon as they answered for the first time. Furthermore, none of them called out the misleading comments at any point in the experiment. When discussing the experiment after it was over, some expert coders claimed that they never read comments, even though they had just responded to the questions in accordance with the comments.

Table 1

Question	Proportion of Coders that were Mislead	
	Experts	**Novices**
1	100.000%	57.143%
2	71.429%	28.571%
3[a]	N/A	N/A
4	85.714%	28.571%

a. this question was not misleading

Table 1 shows the percentage of participants who answered each question according to the comments. The majority of experts followed the cues given by the comments every time, whereas most of the novices did not. It is important to note that this reflects the percentage of coders who based their answer on the comments, not whether they got the question wrong. Some participants answered incorrectly but without using the comments.

When deciding whether a programmer based their answer on the comments or not, we considered their verbal responses. Our misleading comments were either the opposite of what the program was doing (figure 1) or contained unrelated keywords such as "checking for prime numbers" that were in fact not pertinent to the function of the code (figure 2). In the first case, if the participant said the program prints a list of even numbers, we determined that they were misled by the comment. In the second case, if the programmer used these spurious keywords in their answer, it suggested that they were basing their response on the comments. This is because the keywords were chosen to be orthogonal to the true function of the code being presented. Therefore, there was no reason that the program should have triggered these

```java
public class Class2 {
    public static void main(String[] args) {
        //define the limit
        int limit = 50;
        //prints all even numbers from 1 to the limit
        for(int i=1; i <= limit; i++) {
            //if the number is divisible by 2 then it is even
            if(i % 2 != 0) {
                System.out.print(i + " ");
            }
        }
    }
}
```

Figure 1: Example of a misleading comment being the opposite of the truth

```java
public class Class1 {
    public static void main(String[] args) {
        int i = 1, n = 10, t1 = 0, t2 = 1;
        System.out.print(n);
        //condition for sum to be the sum of two prime numbers
        while (i <= n) {
            System.out.print(t1 + " + ");
            int sum = t1 + t2;
            t1 = t2;
            t2 = sum;
            i++;
        }
    }
}
```

Figure 2: Example of a misleading comment being completely different than the actual code

responses.

Our observations are further supported by looking at a series of statistics from our participants. When looking at the average proportion of time spent on comments versus the "meat" of the code for each of the two groups, we can see a big difference (Table 2). In all of the questions, experts spent as much as 13.9% more time on comments than novices. This is a big difference, especially when considering that experts spent less time, on average, on every question. When looking at the average over all questions, experts spent over 7% more time reading comments than novices. It is notable to point out that although most novices did not fall for the comments after the first question, they still read the comments in every question. This could be because they were curious to see if they were correct about them being wrong.

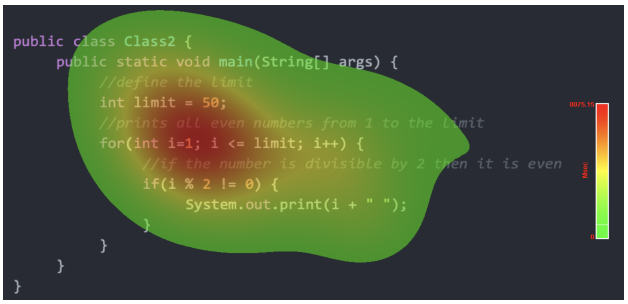
The emphasis put on comments by our experienced participants is further visualized by the aggregate heat maps produced. Using EyeLink DataViewer, we produced heat maps for the images inspected by each of the subjects. To create these heat maps, we applied a two-dimensional Gaussian distribution to each of the fixations. The center is the location of the fixation. The width is regulated by a sigma value of degrees of visual angle, making it so that the area affected by the fixation increases as the sigma value increases. The duration of the individual fixations then

Table 2

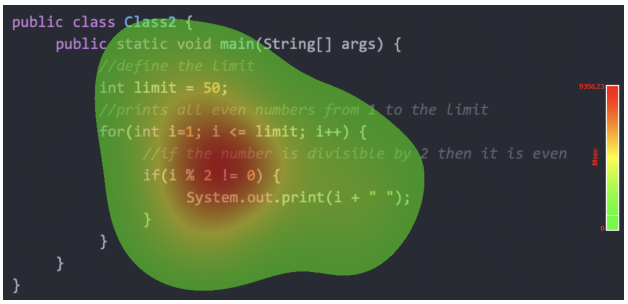
Question	Relative Time Spent on Comments		
	Experts	Novices	Difference
1	63.519%	49.541%	13.978%
2	27.057%	26.936%	0.121%
3[a]	26.090%	15.242%	10.848%
4	17.461%	13.687%	3.774%
Average	33.532%	26.352%	7.180%

a. this question was not misleading

weighs the height of the Gaussian. This 2D Gaussian is added to an internal map by adding weight to that area of the map. This process is then applied to all fixation points and is normalized.



Experts



Novices

Figure 3: Heat map showing where experts (top) and novices (bottom) spent the most time looking for question 1

Figure 3 is the aggregate heat map for experienced and novice coders for question 1. In this graph, it is quite apparent that the experts paid significant attention to the comment while paying little to no attention to the core of the program. This bias toward the comment becomes more apparent when we compare this map to the aggregate map for

the novice coders. In the novices map, we can appreciate the attention that the subjects put on the source code as opposed to the comments. The experts clearly read all three comments and came up with their answer, whereas the novices read the whole snippet but paid more attention to the functioning parts of the code. Since these heat maps are for question 1, the novices had not discovered that the comments were misleading. If we look at the aggregate maps for question 2, we see an even bigger difference in the visual pattern, although the proportion of time spent does not differ much between the two groups.

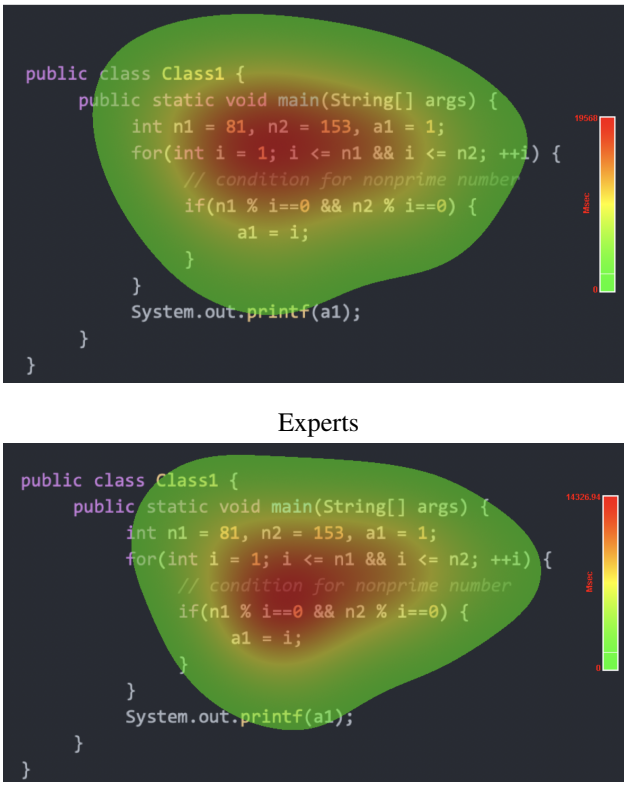


Figure 4: Heat map showing where experts (top) and novices (bottom) spent the most time looking for question 2

Figure 4 is the aggregate heat map for the experts and novices for question 2. Here, it is clear that the experts read the comment and scanned the code whereas the novices spent most of their time on the actual code. Further, in this specific question, it appears the experts read the first few lines of code and the comment, and then they felt confident enough to answer without analyzing the rest of the code. The most common answer given by expert coders for this question was "this program gives you every non-prime number between n1 = 81 and n2= 153". Taking a look at the code it is obvious that this program is not checking whether a number is prime or not. It is calculating the greatest common denominator of n1 = 81 and n2= 153. We determined if a coder was misled by the comment if their answer contained either of the key words "prime" or "non-prime".

These results, while based on a small sample, are both important and surprising. Before starting the experiment, we hypothesized that the experts would look at the short code snippets and would be able to easily determine their functionality. We also thought that novices would be more inclined to look at the comments because they are less comfortable with coding in general and would want the help and insights that comments typically provide. Our observations indicated the contrary. These results are evidence of the emphasis put on comments in a work environment. Although all students are taught that comments are important in their computer science classes, this importance does not necessarily materialize in school-based activities centered on small-scale programming tasks. When in school, students usually only look at and work on their own code. In this style of working, code comments are not imperative to integration activities. Comments become critical in industry, however, where developers are expected to work on code that they did not write, or code that they themselves wrote a long time before. Therefore, it makes sense that professional developers would be more prone to look at comments to guide their answer. This inclination to look at comments is a good thing, so long as the comments are accurate and up to date. Our results emphasize the importance of updating comments when editing code, and to write accurate comments when programming, as developers are quick to trust comments that are technically unsound.

4. Related Works

Eye-tracking has been extensively used in modeling how programmers visually process source code as part of software development activities. In a survey paper exploring 63 studies published between 1990 and 2017 regarding eye-tracking in programming, researchers found that the majority of eye-tracking studies could be categorized into five general topics: code/program comprehension, debugging, non-code comprehension, collaboration, and traceability [9]. Additionally, researchers uncovered a pattern showing that many of the published studies using eye-tracking are based on the same concepts being retested with new data. Our research incorporates each of these five themes, while also introducing a new topic of inline comments.

A similar paper analyzed the various approaches to conducting and using eye movement data in the context of source code [5, 6]. When designing studies using eye movement data, researchers concluded that methods yielding qualitative data were just as important as methods yielding quantitative data. Our research supports similar method-

ologies discussed in the paper by the utilization of eye-tracking data as well as vocal recordings of each participant's thought process. Another example in the research compared expert and novice programmers and how linearly they read source code [3]. The study found that while natural language is read linearly, novice programmers read the source code less linearly, and expert programmers even less linearly. Similarly, our research compared expert and novice programmers, mainly focusing on how susceptible programmers are to misleading comments.

Previously, researchers found that programmers reviewing code skim the file until a snippet of code prompted them to slow down and review a section more thoroughly. These triggers included identifiers, inconsistent code changes, and other confusing or incorrect code [2]. They also found a correlation between the time of the initial scan of a program and the efficiency of identifying errors within code reviews [13]. The researchers concluded that programmers are often not thorough when reviewing code. Their findings show that the eye tends to follow the source code left to right, scan for methods, and jump around to keywords [11]. Additionally, many programmers read about 73% of the code within the first 30% of the review time. Those who took more time scanning were able to identify the errors more efficiently [13].

Another study found that the way a programmer read code appeared to be done in a sequence of patterns such as scanning, jumping ahead and back to look and verify details [8]. This pattern is similar to how individuals read natural text, with the exception that some programmers parse code bottom to top opposed from top to bottom [11]. A workshop analyzing how novices comprehend code also concluded that the participants would approach the code as natural text at first but then utilize more sophisticated patterns as they become more familiar with Java [7].

Moreover, a study comparing differences in reading code found that programmers were more likely to read from top to bottom when they had formed a hypothesis about the code and its function and from bottom to top when they had no understanding of the code and needed to sift through it. The expert programmers tended to focus more on a broader block of code while the novice programmers read line by line [1]. This is corroborated by our study where novices read the whole code snippet, including comments, even when they knew they were misleading. Opposed to our experts that seemed to answer after reading a few lines and the comments. In another study focused on analyzing eye movement when reading Java code, researchers found that the lines with a method call, an if statement, or a variable were fixated on the most [10]. Moreover, attention was focused on the understanding of operators, keywords, and literals, while minimal time was spent on separators [4].

Researchers have been looking for a way to analyze the cognitive processes of developers while they interact with software artifacts. Eye-tracking allows for the visualization of gaze pathways as programmers review code, but it lacks a way to map physiological data to the source code. The developers of VITALISE created a Javascript program that allows researchers the ability to record biometric data from EEG, fMRI, fNIRS, and other measurement devices, mapping it against data recorded in the form of heatmaps from an eye-tracking device. This application of neuroimaging opens doors to understanding the cognitive load on developers as they program [12].

5. Future Work

Due to the nature of the study, some limitations should be noted. First, we used a convenience sample size of both experienced and novice coders from the same institution. It is possible that this sample is not representative of the entirety of the target population and cannot be generalized to other subject groups. Second, while the experiment was conducted in a uniform manner across all 14 subjects, the reality is that the small sample size is a restriction on the certainty with which we are able to say that experienced coders visually pay more attention to comments while reading code snippets. Although we saw a statistically significant difference in the percentage of experienced coders and novice coders who were mislead by the comments, a larger sample size will be necessary to further validate the preliminary results reported here. These limitations are motivation for continuing work in this area.

The results we obtained from this experiment have left the door open for future adaptations and extensions of this study. Using short, simple functions with single in-line comments, we found that experienced coders have a tendency to place a large amount of trust in comments, while novice coders rely more on code. Would these results, however, remain the same if we changed the scope of the code or the commenting style?

It could be interesting to investigate if the programmers would utilize the comments in the same way if the scope of the code changed from the short functions we used in this experiment to longer, more complex programs. For instance, it is possible that we would see that the same novice coders who relied on the code in this experiment might begin to rely more and more on comments as the code shifts from straight-forward to increasingly more complex. How much time and effort needs to be put into reading code before a coder who usually relies on code alone decides to put their trust into comments? A future study revealing whether there is a point in which the length or complexity of a program changes the way a programmer utilizes comments may produce informative results.

The scope of a program, however, may not be the only

factor worth investigating. It could also be noteworthy to see if commenting style plays a role in the trust programmers place in comments. Would our experiment have produced the same results if we used descriptive block comments at the top of each function instead of single, in-line comments? Future works could lead us to exploring if commenting style is simply a preference and that comments are ultimately utilized in the same way, by the same people, or if it affects who reads them and the extent of which they are trusted.

Finally, it may also be interesting to look into how aware coders are of how they use comments. If at the conclusion of this study, for example, we asked each participant about how important comments are to them and how heavily they rely on them, would their responses align with how they performed? Perhaps experienced coders are so familiar with reading code and comments that, especially when the code is fairly straight forward, it becomes somewhat of a mindless activity and they are not fully aware of the extent to which they are using them. This would be yet another interesting adaptation of this study which could provide more information into why we obtained the results that we did.

6. Conclusion

We conducted a small study to gain insight into how programmers read code. In particular, we aimed to find how much a coder relies on and, ultimately, trusts comments, and whether this behavior is related to experience. Using a research-grade, we found that experienced coders in our sample were quick to base their knowledge of a program according to the accompanying comments, even when the comments are wrong. In some cases, they appeared to neglect looking at source code completely. Novice coders, on the other hand, were more likely to accurately describe a program, as they prioritised scanning through and reading the source code, even if they had previously read the comments. Our experiment, while small, has shown that experienced coders feel more confident in relying on comments when working through a program comprehension task, while novice coders tend to be more cautious and use comments in careful conjuction with the source code itself.

References

[1] N. J. Abid, J. I. Maletic, and B. Sharif. Using developer eye movements to externalize the mental model used in code summarization tasks. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pages 1–9, 2019.

[2] A. Begel and H. Vrzakova. Eye movements in code review. In *Proceedings of the Workshop on Eye Movements in Programming*, pages 1–5, 2018.

[3] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm. Eye movements in code reading: Relaxing the linear order. In *2015 IEEE 23rd International Conference on Program Comprehension*, pages 255–265. IEEE, 2015.

[4] T. Busjahn, R. Bednarik, and C. Schulte. 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*, pages 335–338, 2014.

[5] T. Busjahn, C. Schulte, and E. Kropp. Developing coding schemes for program comprehension using eye movements. In *PPIG*, page 15, 2014.

[6] T. Busjahn, C. Schulte, B. Sharif, A. Begel, M. Hansen, R. Bednarik, P. Orlov, P. Ihantola, G. Shchekotova, and M. Antropova. Eye tracking in computing education. In *Proceedings of the tenth annual conference on International computing education research*, pages 3–10, 2014.

[7] T. Busjahn, C. Schulte, S. Tamm, and R. Bednarik. Eye movements in programming education ii: Analyzing the novice's gaze. 2015.

[8] A. Jbara and D. G. Feitelson. How programmers read regular code: a controlled experiment using eye tracking. *Empirical software engineering*, 22(3):1440–1477, 2017.

[9] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng. A survey on the usage of eye-tracking in computer programming. *ACM Comput. Surv.*, 51(1), Jan. 2018.

[10] C. S. Peterson, N. J. Abid, C. A. Bryant, J. I. Maletic, and B. Sharif. Factors influencing dwell time during source code reading: a large-scale replication experiment. In *Proceedings of the 11th ACM Symposium on Eye Tracking Research & Applications*, pages 1–4, 2019.

[11] P. Rodeghero and C. McMillan. 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)*, pages 1–10. IEEE, 2015.

[12] D. Roy, S. Fakhoury, and V. Arnaoudova. Vitalse: Visualizing eye tracking and biometric data.

[13] 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*, pages 133–140, 2006.