Empirical Software Engineering (2024) 29:160
https://doi.org/10.1007/s10664-024-10532-x

An eye tracking study assessing source code readability rules for program comprehension

Kang-il Park[1] · Jack Johnson[2] · Cole S. Peterson[1] · Nishitha Yedla[3] · Isaac Baysinger[1] · Jairo Aponte[4] · Bonita Sharif[1]

Accepted: 22 July 2024

Abstract

Context While developing software, developers must first read and understand source code in order to work on change requests such as bug fixes or feature additions. The easier it is for them to understand what the code does, the faster they can get to working on change tasks. Source code is meant to be consumed by humans, and hence, the human factor of how readable the code is plays an important role. During the past decade, software engineering researchers have used eye trackers to see how developers comprehend code. The eye tracker enables us to see exactly what parts of the code the developer is reading (and for how long) in an objective manner without prompting them.

Objective In this paper, we leverage eye tracking technology to replicate a prior online questionnaire-based controlled experiment (Johnson et al. 2019) to determine the visual effort needed to read code presented in different readability rule styles. As in the prior study, we assess two readability rules - minimize nesting and avoid do-while loops. Each rule is evaluated on code snippets that are correct and incorrect with respect to a requirement.

Method This study was conducted in a lab setting with the Tobii X-60 eye tracker where each of the 46 participants. 21 undergraduate students, 24 graduate students, and 6 professional developers (part-time or full-time)) participated and were given eight Java methods from a total set of 32 Java methods in four categories: ones that follow/do not follow the readability rule and that are correct/incorrect. After reading each code snippet, they were asked to answer a multiple-choice comprehension question about the code and some questions related to logical correctness and confidence. In addition to comparing the time and accuracy of answering the questions with the prior study, we also report on the visual effort of completing the tasks via gaze-based metrics.

Results The results of this study concur with the online study, in that following the minimize nesting rule showed higher confidence (14.8%) decreased time spent reading programming tasks (7.1%), and decreased accuracy in finding bugs (5.4%). However, the decrease in accuracy was not significant. For method analysis tasks showing one Java method at a time, participants spent proportionally less time fixating on code lines (9.9%) and had fewer fixations on code lines (3.5%) when a snippet is not following the minimize-nesting rule. However, the opposite is true when the snippet is logically incorrect (3.4% and 3.9%, respectively), regardless of whether the rule was followed. The avoid do-while rule, however, did not have as significant of an effect. Following the avoid do-while rule did result in higher

Communicated by: Janet Siegmund

Extended author information available on the last page of the article

accuracy in task performance albeit with lower fixation counts. We also note a lower rate for a majority of the gaze-based linearity metrics on the rule-breaking code snippet when the rule-following and rule-breaking code snippets are displayed side-by-side.

Conclusions The results of this study show strong support for the use of the minimize nesting rule. All participants considered the minimize nesting rule to be important and considered the avoid do-while rule to be less important. This was despite the results showing that the participants were more accurate when the avoid do-while rule was followed. Overall, participants ranked the snippets following the readability rules to be higher than the snippets that do not follow the rules. We discuss the implications of these results for advancing the state of the art for reducing visual effort and cognitive load in code readability research.

Keywords Eye tracking · Source code readability rules · Program comprehension · Correctness · Controlled experiment

1 Introduction

Developers spend a lot of time reading and navigating code in order to comprehend its functionality (Minelli et al. 2015). Besides being functionally accurate, the importance of code that is easy to read should not be underestimated. Code that is easier to read leads to getting to the actual task (Alaboudi and LaToza 2021) at hand, such as fixing a bug, much faster. Code readability (Fakhoury et al. 2019) is an important part of program comprehension (Storey 2005; Brooks 1983) for various stakeholders - not only for the person who wrote the code but also for people who will eventually need to read the code to make their own changes to the codebase. Code that is easier to read can have far-reaching effects in terms of the cost of maintaining software in the long run as well as with onboarding new employees on a team. But what do we mean by code readability? What factors affect code readability? These questions could have several possible answers based on individual preferences or even the task at hand. They have also been the source of much debate as to what attributes of code make it more readable.

Text readability models such as the Automated Readability Index (Smith and Kincaid 1970) and Flesch-Kincaid readability tests (Flesch 1948) do not work well with code since code is inherently structured differently from natural text and contains domain-specific syntax and structures. To deal with this issue, Buse and Weimer were one of the first researchers to evaluate code readability in 2008 (Buse and Weimer 2008, 2010) by collecting data from human annotators and deriving associations between code features and developer-perceived readability. Since then, several state-of-the-art readability models (Scalabrino et al. 2016, 2018; Posnett et al. 2011; Daka et al. 2015) have been proposed. Dorn's work (Dorn 2012) further generalized the software readability model to multiple programming languages. However, they are not able to fully capture readability improvements to code (Fakhoury et al. 2019; Mannan et al. 2018). Furthermore, Scalabrino et al. showed that neither readability metrics (text coherence and indentation) nor complexity metrics were correlated with understandability (Scalabrino et al. 2021). Complexity metrics such as McCabe's complexity do not fully capture a developer's perceived readability either (Jbara et al. 2012; Lakshmanan et al. 1991; Gill and Kemerer 1991).

In a reflections article (Posnett et al. 2021) on their original paper of 2011 (Posnett et al. 2011), Posnett et al. point out the distinction between readability and program comprehension and while the research area is actively worked on, not all work in readability is related to

comprehension. Scalabrino et al. for example investigate program comprehension with the code readability aspect (Scalabrino et al. 2016, 2017). A survey done by Santos et al. on how developers perceive code readability based on a set of Java coding practices shows that not all practices actually improved readability (dos Santos and Gerosa 2018).

The question of what makes code readable still remains. How do we define code readability and more importantly, how do we verify if code is readable or not? We believe since code is meant to be consumed by developers, the developer needs to be in the loop when the readability of code is examined. After a meta-analysis, Boswell and Foucher (Boswell and Foucher 2011) present a set of code readability rules (such as simplifying boolean expressions using De Morgan's Law among others) that, if followed, make the code more readable. We direct the reader to Boswell and Foucher (2011) for a complete list of rules. Boswell points out that readability is important for code understandability.

In order to study code readability rules from a developer perspective and to empirically validate Boswell and Foucher's guidelines, Johnson et al. conducted an empirical study (Johnson et al. 2019) to assess two source code readability rules (R1: minimize nesting and R2: avoid do-while). They found that minimizing nesting decreases the time a developer spends reading and comprehending code and also improves their ability to find bugs. However, avoiding the do-while statement did not show a significant impact. Since their demographic was split between native English speakers and non-native speakers, they also show that a better knowledge of the English language led to better readability and comprehension overall.

The study presented in this paper extends the work by Johnson et al. (2019) by replicating the study with an eye tracker (a new mode of data collection) to gather fine-grained insights into how developers read and navigate the code while they are determining the logical correctness of the code. Eye tracking (Obaidellah et al. 2018; Sharafi et al. 2015) equipment is used to measure reading behavior and use that information to determine visual effort spent during comprehension of code that follows and does not follow two readability rules, and examine what specific parts of the code are actually being read and for how long. All participants got both correct and incorrect versions of the code. The main motivation behind conducting this study with eye tracking equipment is to uncover the reading behavior of developers while they are reading and solving comprehension (logical correctness) problems. Such behavior is impossible to uncover via an online questionnaire. In addition to the comprehension tasks, the users were also asked to compare the readability of code snippets that follow or do not follow a rule. The study presented in this paper complements and further explains the results uncovered in the original study by Johnson et al. (2019).

The paper makes the following contributions.

- An eye tracking study is conducted to determine reading differences for two readability rules - minimize nesting and avoid do-while.
- A combination of comprehension and eye movement measures are investigated for each rule.
- A study design setup that includes both correct and incorrect versions of the code is investigated with respect to whether or not they follow the two readability rules.
- A visual effort comparative analysis on how developers rate code on correct solutions in two readability rules.
- A complete replication package with tasks and eye tracking data is provided for verifiability.

Overall, many of the results presented in Johnson et al. (2019) are replicated in the eye tracking study as well. Participants in both studies rated the minimize nesting rule to be much more important than the do-while rule. In addition, the effects of a rule-breaking

snippet and a logically incorrect snippet on task time, self-reported confidence, question correctness, readability assessment, and rating side-by-side method comparison tasks were largely identical with few differences.

With respect to the eye tracking metrics, not following rule R1 (minimize nesting) resulted in fewer navigation around the code, while a logically incorrect snippet resulted in more navigation. For rule R2 (avoid do-while), snippets that do not follow the rule had more fixations with less fixation durations per token, but do not have any effect on how the code was navigated. Logically incorrect snippets also had similarly few effects with only more fixations.

Across both rules R1 and R2, participants ranked the readability of code snippets following the rules to be higher than the snippets that do not follow them.

The paper is organized as follows. The research questions and hypotheses are presented formally in Section 2. Section 3 presents related work in the area of code readability and eye tracking studies in program comprehension. An overview of the original online questionnaire-based study is given in Section 4, which forms the basis of our current eye tracking study. Section 5 describes the experimental design of the eye tracking study presented in this paper. The tasks, stimuli, and group design used is common between both the online questionnaire-based study and the eye tracking study. Detailed experimental results are provided in Section 6. Threats to validity are outlined in Section 7. Section 8 presents the discussion and implications of our work to software engineering education and practice and how eye tracking fits into evaluating code readability. Section 9 concludes the paper by highlighting the contributions and paving the way for future work.

2 Research Questions and Hypotheses

An overview of the experiment goal and factors is given in Table 1. The two main experimental factors (independent variables) are the logical correctness of the code and the rule following with respect to two readability rules namely, R1 (minimize nesting) and R2 (avoid do-while). In the pursuit of quantifying code readability with respect to program comprehension, we measure as part of our dependent variables, the comprehension time (in seconds), comprehension confidence (a 5-point Likert scale), level of understanding (a multiple-choice question and binary-choice question), and readability rating (a binary-choice question in side-by-side method comparison tasks). In addition, the eye tracker allows us to measure specific eye-tracking variables such as fixation counts and fixation durations on the code lines (Sharafi et al. 2015). We also look at derived eye-tracking metrics such as the linear-

Table 1 Experiment Goal and Factors Overview

Goal	Analyze two readability rules using eye-tracking equipment for the purpose of providing empirical evidence of their impact on the understandability of source code.
Experimental factors	Logical correctness, Rule following
Dependent variables	Comprehension Time (Seconds), Comprehension Confidence (5-point Likert Scale), Level of Understanding (Multiple-Choice Question, Binary Choice), Readability Rating (Binary choice Left/Right), Eye Tracking Measures (Fixation Counts, Fixation Durations, Eye Tracking Linearity Measures (Busjahn et al. 2015)).

ity measures (Busjahn et al. 2015) of how the participants read the code at the line-level, quantifying code navigation behavior.

The research questions we seek to address are:

RQ1a Does the minimize nesting rule reduce the *time* a developer spends in understanding logically correct and incorrect source code?
RQ1b Does the minimize nesting rule increase the *level of confidence* a developer has about their own understanding of logically correct and incorrect source code?
RQ1c Does the minimize nesting rule improve the developer's *level of understanding* when reading logically correct and incorrect source code?
RQ2a Does the avoid do-while rule reduce the *time* a developer spends in understanding logically correct and incorrect source code?
RQ2b Does the avoid do-while rule increase the *level of confidence* a developer has about their own understanding of logically correct and incorrect source code?
RQ2c Does the avoid do-while rule improve the developer's *level of understanding* when reading logically correct and incorrect source code?
RQ3 What is the difference in eye movement behavior on logically correct and incorrect source code following and not following the minimize nesting rule?
RQ4 What is the difference in eye movement behavior on logically correct and incorrect source code following and not following the avoid do-while rule?
RQ5 Does eye movement behavior on a snippet align with a preference for following the rule while rating logically correct source code readability rules side by side?
RQ6 Do secondary factors such as native language, Java knowledge, and English knowledge have an effect on accuracy, readability, comprehension, read, and answer time for the tasks?

The first two research questions (RQ1 and RQ2) are a direct replication from the Johnson et al. study (Johnson et al. 2019). RQ1 is related to testing the minimize nesting rule and is split into three sub-questions related to time taken (1a), level of confidence (1b), and level of understanding (1c). RQ2 is related to testing the avoid do-while loop and is similarly split into three sub-questions.

The third and fourth research questions (RQ3 and RQ4) use the fixation counts and durations as well as linearity measures (Busjahn et al. 2015) to assess how participants read each line of code and reports on differences between both correct and incorrect versions of the code for the minimize nesting rule and avoid do-while rule respectively. We refer to all of these metrics together as eye movement behavior. The fifth research question (RQ5) reports on the preference rating of two pairs of correct code snippets shown side by side where one follows the rule and one does not. Note that logically incorrect versions are not used for the side by side comparison. Because it was a rating task, only correct answers are compared. Eye tracking data was collected during this rating preference. The question seeks to determine if there is any correlation between the rating preference and how they read (eye behavior) each of the code snippets (while rating) when shown side by side. The research questions RQ3 through RQ5 are new to this paper and are related to the eye tracking metrics collected using the eye tracking equipment. Based on the research questions above, Tables 2 and 3 list the null and alternative hypotheses for each of the rules, minimize nesting (R1) and avoid do-while (R2), respectively. Note The last research question (RQ6) seeks to determine if secondary factors such as native language, Java knowledge, and English language affect the dependent variables common to both the original and this study. This research question is

Table 2 Null and Alternative Hypotheses for R1 - minimize nesting rule

Null Hypothesis	Alternative Hypothesis
$H1_0$: The use of the minimize nesting rule does not produce a significant reduction on the time a developer spends understanding source code.	$H1_a$: The use of the minimize nesting rule produces a significant reduction on the time a developer spends understanding source code.
$H2_0$: The use of the minimize nesting rule does not produce a significant increase in the level of confidence a developer has about their own understanding of source code.	$H2_a$: The use of the minimize nesting rule produces a significant increase in the level of confidence a developer has about their own understanding of source code.
$H3_0$: The use of the minimize nesting rule does not produce a significant improvement on the level of understanding a developer reaches when reading source code.	$H3_a$: The use of the minimize nesting rule produces a significant improvement on the level of understanding a developer reaches when reading source code.
$H4_0$: The use of the minimize nesting rule does not produce a significant difference in eye movement reading behavior on the source code.	$H4_a$: The use of the minimize nesting rule produces a significant difference in eye movement reading behavior on the source code.

exploratory in nature and did not have a specific hypothesis. In the original study, this analysis was conducted as a secondary factors analysis and not part of the RQs.

3 Related Work

Readable code plays a big role in efficiently maintaining and evolving source code. The use of eye tracking technology in empirical studies lends itself well to understanding the readability of source code. In this section, we present studies and reflections on readability, program comprehension, and the use of eye tracking technology, within the context of developers reading and understanding short code snippets as well as larger codebases.

Table 3 Null and Alternative Hypotheses for R2 - avoid do/while rule

Null Hypothesis	Alternative Hypothesis
$H5_0$: The use of the avoid do-while rule does not produce a significant reduction on the time a developer spends understanding source code.	$H5_a$: The use of the avoid do-while rule produces a significant reduction on the time a developer spends understanding source code.
$H6_0$: The use of the avoid do-while rule does not produce a significant increase in the level of confidence a developer has about their own understanding of source code.	$H6_a$: The use of the avoid do-while rule produces a significant increase in the level of confidence a developer has about their own understanding of source code.
$H7_0$: The use of the avoid do-while rule does not produce a significant improvement on the level of understanding a developer reaches when reading source code.	$H7_a$: The use of the avoid do-while rule produces a significant improvement on the level of understanding a developer reaches when reading source code.
$H8_0$: The use of the avoid do-while rule does not produce a significant difference in eye movement reading behavior on the source code.	$H8_a$: The use of the avoid do-while rule produces a significant difference in eye movement reading behavior on the source code.

3.1 Code Readability Studies

The construction and maintenance of software systems require the coordinated work of many people over long periods of time. This is mainly due to the large size of these systems, the crucial role they play for the organizations they support, and their long life spans. Several studies have examined the type of activities that developers perform when creating and maintaining software, with the aim of categorizing them and determining how much effort they require. Their findings indicate that developers spend much of their time and effort reading and understanding code written by others (Minelli et al. 2015; Xia et al. 2018). The main reason is that understanding code is an essential activity to successfully perform typical tasks such as reusing, testing, correcting, and extending existing code. Therefore, from an economic standpoint, increased code readability could significantly reduce the time and cost of developing and maintaining software systems.

From a research perspective, code readability has been studied from different angles. (i) developers' perception of readability has been characterized (Sedano 2016; dos Santos and Gerosa 2018), (ii) the impact of a wide range of coding practices on readability has been analyzed (Ajami and Woodbridge 2017), and (iii) various metrics and theoretical models have been proposed to characterize and automatically assess code readability (Scalabrino et al. 2018).

Practitioners usually describe readable code as one that clearly reveals its logic and communicates its intent to the reader. However, they sometimes disagree on the specifics that determine whether a code is readable or not. From an online survey by Wiese et al. (2019), novice programmers perceive code written by experts as less readable than code written by them, although this does not seem to affect their degree of comprehension of both types. Sedano (2016) reports a study where readers and authors of code follow a process to analyze and assess readability. Repeated use of this process allowed both groups to improve their abilities to write readable code. In addition, the study reports improving naming, reducing code duplication, and simplifying code structures as the most prevalent strategies to improve readability. Hunter-Zinck et al. propose rules for writing clean code and designing tests, drawn from their experiences as developers for industry, academia, and government laboratories (Hunter-Zinck et al. 2021). Ljung et al. report that developers mostly agree with clean code principles (Martin 2009) and believe that they improve readability, reusability, and maintainability (Ljung and Gonzalez-Huerta 2022). In a study that interviewed students, educators, and developers, Börstler et al. found that all three groups consider readability as the most important aspect of code quality, above facets such as structure, understandability, security, and testability (Börstler et al. 2018). Piantadosi et al., based on the results of an empirical study where the evolution of 25 open source projects was analyzed, propose guidelines and programming practices to keep a code base readable over time (Piantadosi et al. 2020). The research about the factors that influence the readability of test code has identified naming, comments, test summaries, and test structures as the most relevant ones (Winkler et al. 2022).

Many researchers have examined a wide variety of factors that could potentially affect readability. Some research has shown that the impact of a name on readability depends not only on how good or bad it is, but other factors such as the role it plays within the code, where it appears, its length, and its type can also play a role (Avidan and Feitelson 2017; Schankin et al. 2018; Cates et al. 2021). Apart from naming, many other coding practices have been analyzed. dos Santos and Gerosa (2018) found out that 7 out of 11 Java coding practices improve the readability perceived by developers, while one of them decreases readability and the others are neutral. The assessed practices refer to lexical and syntactic features or algorithmic simplicity and were derived from existing code readability models (Buse and

Weimer 2010; Scalabrino et al. 2016). Mi et al. (2023) used a causal analysis process on 420 labeled code snippets to find that a higher number of comments increase code readability while more assignments, identifiers, and periods decrease code readability. Börstler et al. (2016) used the length of lexemes and the number of lexemes per block to create a scoring system for readability and determined that the system correlates well with code snippets provided by textbooks. Ajami and Woodbridge (2017) empirically analyzed the effects of syntactic constructs (e.g., conditionals and loops) on the ability of developers to understand the code. They used a game-like online questionnaire with 222 professional progammers, and the results indicated that nested if statements took less time to read vs. an equivalent single if expression but was not statistically significant. Barbosa et al. (2022) compared the readability of pairs of Java code snippets as perceived by professional developers. They also calculate the readability of these pairs using the Posnett's model (Posnett et al. 2011). In each pair, one is the original code snippet, while the other is a refactored version, resulting from applying design techniques aimed at reducing complexity. In most cases, the developers considered the refactored version more readable. By contrast, Posnett's model finds the original versions slightly more readable.

Oliveira et al. (2020) conducted a systematic literature review examining studies with human subjects analyzing how different structural and visual characteristics of source code impact its readability and legibility. They recently conducted another systematic literature review (Oliveira et al. 2023) in which they report on 15 studies conducted with human developers where they observed that appropriate indentation, end block delimiters, and limiting line length to within 80 characters had a positive impact on code readability. However, they also observed that results on identifier styles show divergent results and emphasized there is still a lack of work in determining code readability of source code.

As Posnett et al. pointed out (Posnett et al. 2011), since readability is a code aspect affected by subjectivity, its measurement requires human studies to collect readability scores and perform statistical analyses of inter-rater agreement that allow researchers to distill code metrics and formulate predictive models of readability. For this purpose, there have been progressive improvements in the formulation of metrics and predictive models. Buse and Weimer (2010) identified simple code features, correlated with human notions of readability, and based on them built one of the first measures for this feature. This seminal work materialized the possibility of automating the continuous assessment and monitoring of the readability of source code. Later models have improved performance in different ways. For example, Posnett et al. (2011) introduced size, code entropy, and Halstead metrics into their model. Scalabrino et al. (2016, 2017) showed evidence that considering textual features of code (e.g., terms extracted from identifiers and comments) resulted in a model with higher readability prediction power and significantly higher accuracy as compared with all the other state-of-the-art models. Mi et al. (2018) determined that the use of a human annotator on an inception model increased its classification accuracy relative to other state-of-the-art models. In later work (Mi et al. 2022), they used three different representations of the source code. These representations capture the code's visual, semantic, and structural features, respectively. Besides that, their model includes a neural network that extracts those features and a classifier that ultimately determines whether the code is readable. This work is more artifact-centric than developer-centric. The study done in this paper looks at readability from the perspective of the developer actually reading the code compared to looking at artifact-centric methods derived solely from the code itself, such as entropy by prior work.

A reflection by Posnett et al. (2021) lays out the evolutionary history of readability studies and models, building off of the work of Buse and Weimer (2010). They also point out that the results in readability have led to recent work on assessing the understandability of the

code snippets (Scalabrino et al. 2021). This is because a deep understanding of a piece of code is sometimes influenced not only by the readability of the code but also by factors such as external software documentation and the intrinsic characteristics of the developer.

3.2 Eye Tracking Studies in Readability and Comprehension

Eye tracking provides a useful resource to gain evidence and insight of cognitive processes when analyzing source code. Sharafi et al. (2015) conducted a systematic literature review on 35 relevant eye tracking studies using eye tracking technology in software engineering. Their work identifies limitations of eye-trackers and provides general recommendations for researchers looking to perform eye tracking studies. Obaidellah et al. (2018) presents a systematic mapping study on studies using eye tracking alongside programming experiments. Of the 63 reported studies published after 2011, they found 60% utilize eye tracking technology, with a majority focused on code comprehension and debugging. They reiterate some of the current limitations of eye tracking technology and analyze trends in experimental designs. They state that trends can be due to those limitations.

Several eye tracking studies investigate the effects of identifier naming conventions and indentation conventions on program comprehension. Binkley et al. (2013) presented a family of 5 studies with 150 participants that investigated the impact of identifier style, namely the use of camel case and underscore, on human comprehension. Data collection involved the use of online questionnaires and eye tracking. The results suggest that camel case is easier for novice programmers to read but the differences between camel case and underscore for variable identifiers are minimal for experienced programmers. An online study of 128 participants was presented by Lawrie et al. (2006) that investigated the effects of the level of abbreviation of identifiers, namely single letters, abbreviations, and full words. The study results showed that full word identifiers and its first letter abbreviations had the highest level of comprehension compared to single letter identifiers, but there was no significant difference in comprehension between the former two. Schankin et al. (2018) presented a web-based study of 88 participants that evaluated the effects of longer but more descriptive variable identifiers on code comprehension. The study showed that experienced programmers benefited from the more descriptive identifiers when an in-depth understand of the code was required as they spent less time reading the code and also jumped back and forth less. However, novice programmers and tasks involving finding syntax errors were not affected by this. Bauer et al. (2019) presented a replication study with 22 participants that evaluated the effects of code indentation on program comprehension and visual effort. Their results suggest that indentation is a purely stylistic choice in modern programming languages such as Java, unlike older languages the original study by Miara et al. (1983) used for its tasks such as Pascal.

There are studies that investigate how code is presentated on an IDE effects code readability. The impact of color coding in code syntax was investigated by Beelders and du Plessis (2015). They compared the differences in eye movement between code that is displayed in black-and-white and code with syntax coloring among 34 participants. The results suggested that while students considered the code with syntax coloring to be more readable and the number of fixations, fixation durations, and regressions were lower, there were no statistically significant differences. Park et al. (2023) presented a study where they compared how computer science students read code depending on whether there was scope-based code highlighting within a Java IDE. There were no notable differences between the two code presentation styles, but they did notice less code navigation when using a frame-based language similar to Java.

As a precursor to this larger study, a preliminary analysis of a subset of 14 participants (out of 46) from the dataset was conducted, focusing on the method comparison tasks. The findings from this initial analysis were presented in a short paper at a readability workshop (Peterson et al. 2021). The results show that developers rated the snippet that avoided nested-if statements as more readable with no clear preference for avoid do-while statements. They also reported more fixations on the snippet that avoided do-while loops. A comprehensive analysis on the complete dataset with more insights of code navigation via metrics on linearity patterns is presented in the current paper.

Code readability is a research topic that is gaining a lot of attention in recent years. In summary, our work provides evidence-based eye tracking data to support code readability rules from the developer's reading perspective. We do not claim that readability is the same as comprehension. We believe readability is one factor (among many) that can aid in program comprehension.

4 Previous Online Questionnaire Study on Code Readability

In our prior work (Johnson et al. 2019), we conducted an online questionnaire-based controlled experiment on assessing two code readability rules. We used Qualtrics [1] to test for comprehension differences between rule-following for the minimize nesting rule and the avoid do-while rule. Thirty-two Java methods were tested belonging to one of four categories: a) the Java method followed the minimize nesting/do-while rule and was correct in relation to the specification, b) the Java method did not follow the minimize nesting/do-while rule and was correct in relation to the specification, c) the Java method followed the minimize nesting/do-while rule and was incorrect in relation to the specification, and d) the Java method did not follow the minimize nesting/do-while rule and was incorrect in relation to the specification. This online study was conducted with 275 participants.

The results of this study showed us that following the minimize nesting rule decreases the time a developer spends reading and understanding the code. It also increases the confidence in the developer's understanding of the code and improves their ability to find bugs (i.e., for the ones that were incorrect based on the specification. The same cannot be said about avoiding the do-while rule as no significant impact was found on the various variables of the level of understanding, time spent reading and understanding, confidence in understanding, or the ease of finding bugs. The prior study also found that knowledge of English impacted readability and comprehension confidence when the minimize nesting rule was followed.

Since this study was conducted online, we were not able to see what lines of the code snippet the participants were reading. In the questionnaire-based study, the time spent reading and understanding was based on how long they spent on the Qualtrics page showing the Java snippet as a whole. In order to understand in more detail what the developers were actually reading, we decided to conduct a follow-up study using an eye tracker. Table 4 shows the two different groups that are discussed in this paper. The section on Participants (Section 5.3) describes the eye tracking participants in more detail. The Johnson et al. paper (2019) describes the online study participants in more detail.

The study design i.e., Java methods used, rules used, and groups were the same between the online and eye tracking versions of the study. The only difference was in the data collection mode where the eye tracker was used instead of an online questionnaire to collect data. In

[1] https://www.qualtrics.com

Table 4 Number of participants and mode of data collection for each study

Study	Mode of Data Collection	Number of Participants
Online study by Johnson et al. (2019) (prior study)	Online Questionnaire using Qualtrics	275
Eye tracking study	Eye Tracker (X-60) using Tobii Studio	46

the next section, we describe the experiment design in more detail and mention the parts that are unique to the eye tracking replication.

5 Experimental Design

The *goal* of this study is to *analyze* two code readability rules *for the purpose of* gathering objective empirical evidence (via eye tracking equipment) of their impact on the comprehension of logically correct and incorrect source code. This study extends our prior work (Johnson et al. 2019) using eye tracking equipment. As in Johnson et al. (2019), we analyze two code readability rules - one is designed to make source code control flow easier to read - R1: Minimize nesting. The second rule suggests that while loops are preferred over do-while loops - R2: Avoid do-while loops. The quality focus of these rules is the effectiveness (readers' level of understanding reached), efficiency (reader's time spent), reader's level of confidence about their own understanding of the source code, and eye movement behavior on source code lines where the rules are implemented. The *perspective* is that of a software engineering researcher evaluating code readability rules for comprehension in the *context* of developers reading and understanding short Java code snippets that follow/break the two readability rules that are logically correct and incorrect.

5.1 Study Tasks

We had two categories of tasks: 8 Single Method Analysis Tasks (4 for each rule) and 2 Method Comparison Tasks (1 for each rule). Each Single Method Analysis Task consisted of a problem specification along with a method that implements a possible solution. In the Method Comparison Tasks, two methods were shown side-by-side with one following a rule and the other that does not. In total, 16 Java methods are used to test one readability rule. A participant does not require an understanding of code outside of the code that is immediately presented to them, because the methods do not call other methods or use objects of other classes.

Each readability rule was assigned four basic programming problems as presented in Tables 5 and 6. Every programming problem has four solutions each in the form of Java methods. For the sake of clarity and organization, a method is denoted as $P_k R_j L_i$, where P_k is the problem it solves and R_j is whether the problem follows the readability rule and L_i is whether the problem is logically correct, where $k \in 1..4$ and $i, j \in \{0, 1\}$. The 0 indicates either not following the rule or logically incorrect and 1 indicates following the rule or logically correct. Each of the four solutions to P_k represent treatments T1, T2, T3, and T4 with the following characteristics:

– $P_k R_1 L_1$ is a correct solution to P_k that follows the readability rule we are interested in (Treatment T1).

Table 5 Problems for R1 (Minimize nesting rule)

Problem	Problem Statement
P1	Given three integer numbers, the method must return the greatest.
P2	Given a mark, which is an integer between 0 and 100, the method must return a letter. Letter A if $mark \geq 90$; B if $mark \in [80, 90)$; C if $mark \in [70, 80)$; D if $mark \in [60, 70)$; letter F, otherwise.
P3	Given the body mass index (bmi), the method must return the category in which the index is located. The category is "very severely underweight" if $bmi \leq 15$; "severely underweight" if $bmi \in [15, 16)$ "underweight" if $bmi \in [16, 18.5)$; "healthy weight" if $bmi \in [18.5, 25)$; "overweight" if $bmi \geq 25$.
P4	Given three integers, the method must count how many of them are positive numbers.

- $P_k R_0 L_1$ is a correct solution to P_k that does not follow the readability rule we are interested in (Treatment T2).
- $P_k R_1 L_0$ is an incorrect solution to P_k that follows the readability rule we are interested in (Treatment T3).
- $P_k R_0 L_0$ is an incorrect solution to P_k that does not follow the readability rule we are interested in (Treatment T4).

For the first category of 8 single method analysis tasks, for each rule, the participant was shown the method's functionality in plain English at the top. The code for that problem was then shown below the prompt. See Fig. 1 for an example of a correct solution of Problem 4 for the minimize nesting rule where given three integers, the method must count how many of them are positive numbers, where one code snippet follows the rule ($P_4 R_1 L_1$) and the other does not ($P_4 R_0 L_1$). In the method analysis tasks, the participant was required to read the problem statement given in plain English on top of the page and the code corresponding

Table 6 Problems for R2 (Avoid do-while loops rule)

Problem	Problem Statement
P1	Ask the user to answer a multiple choice question. Show the question, get the user response, and end when the user chooses the correct answer or when she decides not to try more (typing 'q' or 'e')
P2	Let's assume that we want to force a user to change her password. The user must give a new password that must have 4 different characters. Besides, the given password must be different to the old one. The method receives the old password as a parameter and asks the user to give the new one.
P3	Add the positive numbers in an array. The method receives an integer array as a parameter and must return the sum of the positive numbers in the array.
P4	Count the occurrences of a character. This method receives a string and a character as parameters. It must count and return the number of occurrences of the character in the string.

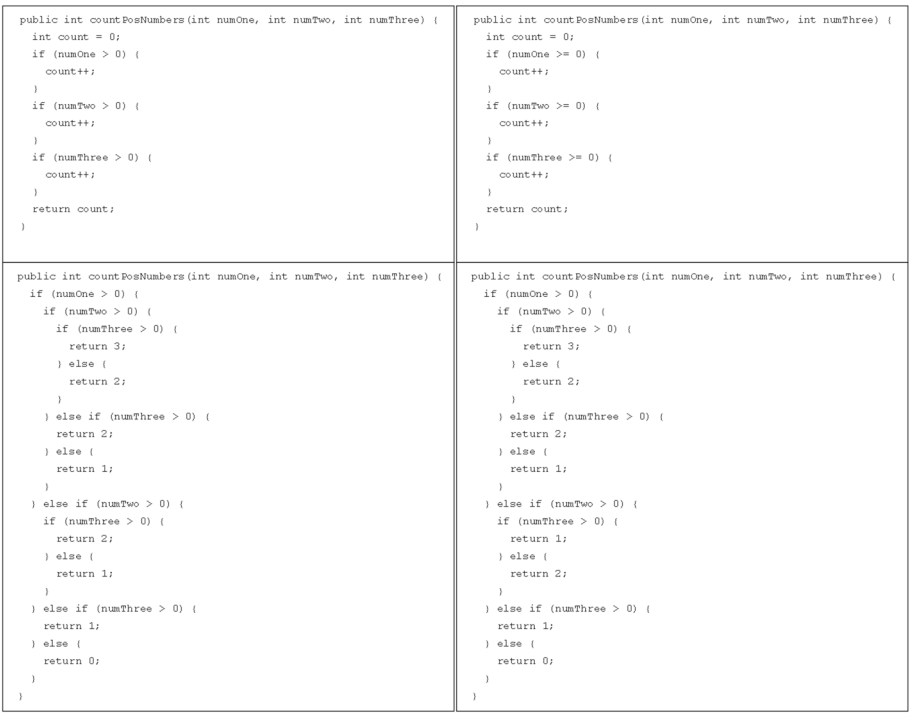

Fig. 1 Four methods for Problem 4 - CountPosNumbers. The method on the top left is correct and follows R1 ($P_4R_1L_1$). The bottom left method is correct but does not follow R1 ($P_4R_0L_1$). The top right method is logically incorrect but follows R1 ($P_4R_1L_0$). The bottom right method is logically incorrect and does not follow R1 ($P_4R_0L_0$) In Problem 4 the requirement was: Given three integers, the method must count how many of them are positive numbers

to that statement following it. After reading they were asked to rate the readability of the method on a 5-point Likert scale, answer a multiple-choice comprehension question, rate confidence of the answer on a 5-point Likert scale, and finally a binary choice on whether the method is logically correct.

For the second category of 2 method comparison tasks, for each rule, the participant was shown two methods side-by-side on the screen (see Fig. 2) where one method was following the rule and the other was not following the rule. The method's functionality was given in plain English at the top. The prompt assigned to them was to analyze the code and choose which method they thought was more readable with an optional comment justifying their choice. The code that followed the rule was always placed on the left but we found that participants transitioned to both code snippets before making their selection. This observation was made systematically by analyzing the fixation graphs and a replay of the sessions in Tobii Studio.

Method features in addition to rule following and logical correctness were measured as shown in Table 7. In summary, for a method analysis task, the participant reads a problem statement, analyzes one of the four methods proposed as a solution for that problem, rates the readability of the method, answers a multiple-choice comprehension question, self-rate confidence of level of comprehension, and determines whether or not the method is correct.

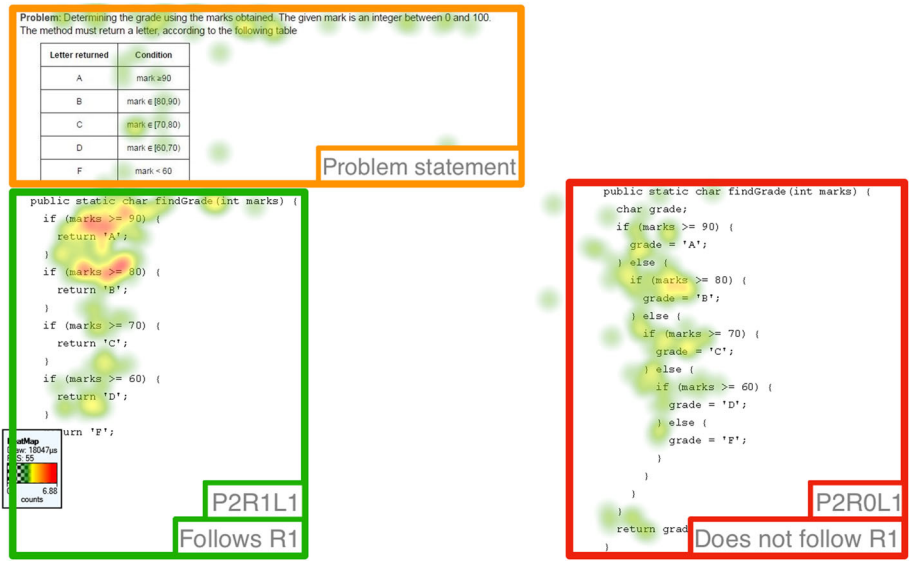

Fig. 2 A screenshot from one participant for R1: minimize nesting with the left ($P_2R_1L_1$) following the rule and the right ($P_2R_0L_1$) not following the rule. The method functionality is given on top. A gaze heatmap is overlayed on top, which illustrates where a participant directs their attention on a screen, with warmer colors denoting a higher concentration of gazes

In a method comparison task, the participant reads problem statement P_k and analyzes the two correct solutions to that problem ($P_kR_1L_1$ and $P_kR_0L_1$). Once the participant analyzes both solutions, she selects the most readable one and provides the rationale behind her choice.

5.2 Eye Tracking Apparatus

The Tobii X60 eye tracker running at 60Hz was used for this study conducted on a 23-inch LCD monitor. The study was conducted using vendor-provided Tobii Studio software to record eye gaze on the code snippets that were displayed on the screen. The drift as reported by the vendor is typically 0.1 degrees and allows a head movement error of 0.2 degrees. A 9-point calibration was used before starting the study. Each Java method was shown in Tobii Studio as an image and fit within the screen limits. Software provided by the vendor was used to map the raw gazes to each line of code (an area of interest - AOI) as fixations by using the I-VT fixation filter (Andersson et al. 2017). For each Java method represented as an image, an area of interest (AOI) was created for every line. In this study, our focus is on line-level data for each Java method.

The code snippets were carefully formatted to facilitate accurate eye tracking. Specifically, they were presented in 14-point Courier New font, single-spaced, with a tab indent of 4 spaces. To minimize confounding factors, syntax highlighting was deliberately omitted. The chosen font size ensured that the X60 eye tracker, with its accuracy of 0.5 degrees, could accurately track each line of code.

Table 7 Code Snippet Metrics for Minimize Nesting (first four problems) and Avoid-do while (last four problems) along with the McCabe Complexity and Participant Grouping

Method Name	Name	Logically Correct?	Follows Rule?	Lines	Characters Per Line	Max Indentation	Average Indentation	Identifiers	Avg Identifier Length	Max Occurrences Of Any Identifier	If Statements	Max Paren Nest Level	Max Conditions Per If Statement	McCabe Complexity	Number of Participants
FindTheBiggest1	$Rule_1P_1R_1L_1$	yes	yes	10	23.4	2	1	4	7.5	6	2	1	1	3	7
FindTheBiggest2	$Rule_1P_1R_0L_1$		no	17	20.06	3	1.71	4	7.5	6	3	1	1	4	7
FindTheBiggest3	$Rule_1P_1R_1L_0$	no	yes	13	26.92	2	1.08	4	7.5	6	3	2	2	7	7
FindTheBiggest4	$Rule_1P_1R_0L_0$		no	14	22.36	3	1.29	4	7.5	5	3	2	2	5	8
FindGrade1	$Rule_1P_2R_1L_1$	yes	yes	15	14.53	2	1.13	1	5	5	4	1	1	5	6
FindGrade2	$Rule_1P_2R_0L_1$		no	21	15.48	5	2.43	2	5	7	4	1	1	5	8
FindGrade3	$Rule_1P_2R_1L_0$	no	yes	15	17.2	2	1.13	1	5	8	4	1	2	8	5
FindGrade4	$Rule_1P_2R_0L_0$		no	15	21.07	2	1.2	2	5	8	4	1	2	8	11
BodyMassIndex1	$Rule_1P_3R_1L_1$	yes	yes	16	19.94	2	1.13	1	3	5	4	1	1	5	7
BodyMassIndex2	$Rule_1P_3R_0L_1$		no	21	20.57	5	2.43	2	4.5	7	4	1	1	5	5
BodyMassIndex3	$Rule_1P_3R_1L_0$	no	yes	16	22.88	2	1.13	1	3	8	4	1	2	8	10
BodyMassIndex4	$Rule_1P_3R_0L_0$		no	21	22.81	5	2.43	2	4.5	8	4	1	2	8	8
CountPosNumbers1	$Rule_1P_4R_1L_1$	yes	yes	13	16.31	2	1.08	4	6.25	5	3	1	1	4	9
CountPosNumbers2	$Rule_1P_4R_0L_1$		no	25	16.88	4	2.08	3	6.67	5	7	1	1	8	10
CountPosNumbers3	$Rule_1P_4R_1L_0$	no	yes	13	16.54	2	1.08	4	6.25	5	3	1	1	4	8
CountPosNumbers4	$Rule_1P_4R_0L_0$		no	25	16.92	4	2.08	3	6.67	5	7	1	1	8	2
WhoIsTheAuthor1	$Rule_2P_1R_1L_1$	yes	yes	27	31.44	3	1.63	3	5.3	5	2	2	2	5	10
WhoIsTheAuthor2	$Rule_2P_1R_0L_1$		no	27	31.3	3	1.59	3	5.3	5	2	2	2	5	6
WhoIsTheAuthor3	$Rule_2P_1R_1L_0$	no	yes	25	33.32	3	1.56	3	5.3	5	2	2	2	5	10
WhoIsTheAuthor4	$Rule_2P_1R_0L_0$		no	26	32.04	3	1.54	3	5.3	5	2	2	2	5	3
ChangePassword1	$Rule_2P_2R_1L_1$	yes	yes	23	25.57	6	2.26	6	3.83	9	2	2	2	7	6
ChangePassword2	$Rule_2P_2R_0L_1$		no	24	24	6	2.21	6	3.83	9	2	2	2	7	9
ChangePassword3	$Rule_2P_2R_1L_0$	no	yes	22	25.82	5	1.95	6	3.83	9	2	2	2	7	4
ChangePassword4	$Rule_2P_2R_0L_0$		no	23	24.17	5	1.91	6	3.83	9	2	2	2	7	10
SumPositiveNums1	$Rule_2P_3R_1L_1$	yes	yes	12	18.58	3	1.25	4	4.75	5	1	1	1	3	3
SumPositiveNums2	$Rule_2P_3R_0L_1$		no	16	16.63	3	1.25	4	4.75	5	2	1	1	3	8
SumPositiveNums3	$Rule_2P_3R_1L_0$	no	yes	12	18.67	3	1.33	4	4.75	5	1	1	1	3	9
SumPositiveNums4	$Rule_2P_3R_0L_0$		no	16	16.69	3	1.31	4	4.75	5	2	1	1	3	9
CountLetter1	$Rule_2P_4R_1L_1$	yes	yes	14	18.93	3	1.43	5	5.4	5	1	2	1	3	10
CountLetter2	$Rule_2P_4R_0L_1$		no	18	17	3	1.39	5	5.4	5	2	2	1	4	6
CountLetter3	$Rule_2P_4R_1L_0$	no	yes	13	19.38	3	1.31	5	5.4	4	1	2	1	3	6
CountLetter4	$Rule_2P_4R_0L_0$		no	17	17.24	3	1.29	5	5.4	4	2	2	1	4	7

Names are in the form of $Rule_lP_kR_jL_i$, where $l \in \{1, 2\}$, $k \in 1...4$, and $j, i \in \{0, 1\}$. $Rule_l$ is the readability rule being evalulated, P_k is the problem it solves, R_j is whether the problem follows the readability rule, and L_i is whether the problem is logically correct

The subscript 1 indicates either the snippet following the rule (R_1) or a logically correct snippet (L_1)

The subscript 0 indicates either the snippet breaking the rule (R_0) or a logically incorrect snippet (L_0).

5.3 Participants

The participants were recruited from Youngstown State University in Youngstown, Ohio, USA. They were given extra credit to participate in the study. The recruitment was done via flyers and word of mouth in CS courses at the university. The study was conducted with 46 participants. Out of the 46, 21 (45.6%) participants were undergraduate students in computer science, 24 (52.2%) were graduate students in computer science, and 6 (13%) were working as professional developers. The majority of the participants (24, or 52.2%) spoke English as their native language while other participants spoke Telugu, Nepali, and Hindi as their native languages. 1 (2.2%) participant spent more than 5 years working as a professional programmer, 8 (17.4%) participants had between 1 and 3 years of professional experience, 11 (23.9%) had less than 1 year of professional experience, and 27 (58.7%) participants did not have professional experience. Most participants (24, or 52.2%) indicated that they spent more than 1 year programming in Java, while a similar number of participants (23, or 50%) indicated that they spent less than a year programming in Java. Two (4.3%) participants self-reported as having very good knowledge of the Java programming language, 16 (34.8%) had good knowledge, 25 (54.3%) had satisfactory knowledge, and 4 (8.7%) identified as having poor knowledge of Java.

5.4 Variables

This section outlines all the independent and dependent variables in this study. The two independent variables in this study are:

- **Logical Correctness:** Indication of whether the Java method satisfies the problem specification
- **Rule Following:** Indication of whether the Java method follows the readability rule being tested

Each row in Table 8 shows a dependent variable, its definition, and which research question it relates to. Eye tracking metrics are highlighted in gray, while the others are identical to the original online study.

5.5 Study Procedure

All the data was collected using Tobii Studio software in a controlled lab setting. All participants participated in the study in the same lab room, and only one participant was present alongside the study moderator at the lab at a time. Each participant was within the optimal range of the eye tracker (50-90cm), and only artificial lights were in the room with minimal noise to the lab, minimizing distractions. When the participant arrived at the lab, they were briefed about the study and process and given instructions on how to be seated in front of the eye tracker. First, they filled out the pre-questionnaire using an online form. The pre-experiment questionnaire collected the participants' demographic information regarding their programming experiences, such as general programming experience, experience in the Java language, and reading skills in English.

After this, the next step was for them to sign the consent form. After the consent was obtained, we then proceeded to calibrate the eye tracker to the participant's eyes while they were seated in front the device. We then perform a 9-point calibration, and if it was successful,

Table 8 Dependent variables used for each of the Research Questions

Research Questions	Metric	Definition
RQ1, RQ2, RQ6	Level of Understanding	Two questions rating the comprehension of a given snippet (Multiple-choice **Comprehension Question**) and whether or not the snippet is logically correct (Binary question on **Logical Correctness**)
RQ1, RQ2, RQ6	Comprehension Time	Time taken to read and answer the task problem (seconds)
RQ1, RQ2, RQ6	Readability Assessment	Participant's perception/ratings of how readable each code snippets is (5-point Likert)
RQ1, RQ2, RQ6	Comprehension Confidence	Rating of how confident participants were about their own understanding level and the correctness of their answers to the comprehension questions (5-point Likert)
RQ1, RQ2, RQ6	Readability Rating	Rating of which snippet is more readable for side-by-side method comparison tasks (Binary question)
RQ3, RQ4, RQ5	Fixation Counts	The number of fixations on an Area of Interest (AOI)
RQ3, RQ4, RQ5	Fixation Durations	The length (in miliseconds) of fixations on an AOI
RQ3, RQ4, RQ5	Vertical Next Text*	The percentage of forward-moving saccades that either stay on the same line or move a line down.
RQ3, RQ4, RQ5	Vertical Later Text*	The percentage of forward-moving saccades that either stay on the same line or move down any number of lines.
RQ3, RQ4, RQ5	Horizontal Later Text*	The percentage of forward-moving saccades within the same line.
RQ3, RQ4, RQ5	Regression Rate*	The percentage of backward-moving saccades.
RQ3, RQ4, RQ5	Linear Regression Rate*	The percentage of backward-moving saccades within the same line.
RQ3, RQ4, RQ5	Line Coverage*	The percentage of lines a participant looked at for each snippet.
RQ3, RQ4, RQ5	Saccade Length*	Average distance between every successive pair of fixations.

Eye tracking metrics are highlighted in gray
The * denotes linearity metrics derived from Busjahn et al. (2015)

they began the study tasks. The study uses a within-subjects design to have participants serve in all treatments. The study questionnaire is divided into parts A and B. In part A, the participant performs 8 method analysis tasks, 4 related to rule R1 and the other 4 related to rule R2. In part B, the participant performs two method comparison tasks, one related to R1 and the other related to R2. Once the participant finishes all study tasks, they fill out a post-experiment questionnaire ranking the importance of R1 and R2 in regards to writing readable code. To avoid the carryover effect confounding results, each participant is given the four analysis tasks for each rule in random order with each task corresponding to a different problem. After they were done with all the tasks, they filled out a post questionnaire where we asked them to rank the importance of the two rules for writing readable code. After the participants left, the data was exported out of Tobii Studio for further correction and analysis.

5.6 Verifiability

The replication package for this study that includes all the tasks, scripts, protocols, questionnaires, and eye tracking data can be found at https://osf.io/m39p8/ (Park et al. 2023).

6 Experimental Results

In this section, we first describe the pre-processing and data correction that was performed on the eye tracking dataset. The rest of the sections present the research questions' results and their corresponding hypotheses introduced in Section 2.

6.1 Data Pre-processing and Correction

After all the data was collected, all fixations (Andersson et al. 2017) were generated for each of the stimuli/code snippets presented to the participants. For this purpose, we used the I-VT fixation filter using the default settings available in Tobii Studio (Olsen 2012). In other words, we had the *max gap length* at 75ms, *window length* at 20ms, *velocity threshold* at 30°/s, *max time* at 75ms, *max angle* at 0.5°, and *minimum fixation duration* of 60ms. Next, all of the fixations for each stimuli were exported from Tobii Studio for data correction. The data was corrected to make sure all the gazes corresponded to the correct line. This correction was performed in two stages. During the first stage, two of the paper authors independently validated and corrected (if necessary) each of the fixations using a custom-built program that displayed 10 fixations at a time overlaid onto the relevant areas of interest of the stimulus and allowed them to move the set of 10 fixations up, down, left, and right to align them with the code lines better. The main areas of interest were the lines of code in the code snippets and the program's text description displayed on the top of the screen. These corrections were stored as two numbers which represented the distance moved in pixels on each axis from the initial recording to the corrected one. This method of correction is common in eye tracking studies (Busjahn et al. 2015) where there is some drift with the data as the participant views a line. A similar correction scheme was done by Busjahn et al. (2015). Note that eye fixations were not cherry-picked and moved one at a time. Instead, the trajectory of the eye movements was followed and only groups of 10 fixations were moved corresponding to the trajectory as shown on the overlay. This initial labeling process took an average of 8 hours for each researcher.

Once each set of individual corrections was complete, they were compared via the use of a script, and any disagreements were recorded. A pair of corrections was only considered to be in disagreement if there were any differences in the words that each fixation landed on between the two corrections, and in the case of a pair that was in agreement, a random correction from the two was chosen to be used for the final dataset.

After the disagreements were compiled, the two researchers met and went through them one at a time to discuss and agree on which correction to use. While discussing disagreements, the researchers could see the initial recorded fixations and each set of corrections, but did not know who had recorded which set of corrections. When comparing the disagreeing label sets, the reviewers chose the one that best matched the fixations to the lines and sections of the codeblocks. This process took about four hours over the course of two meetings. Please

note that the script that was first run to generate the candidate list flagged things solely by pixel value not by actual words the fixation fell on. For single method analysis tasks, 30,798 out of 52,110 fixations were flagged. For side-by-side method comparison tasks, 4,534 out of 13,386 fixations were flagged. This generated a much larger disagreement list than it actually was. However, to be complete and thorough, the two coders went through the entire generated list from the script. They found that most of the disagreements tagged by the script were on the same object. This method of deciding on the final set is much more thorough and validates all the supposedly flagged disagreements by the script.

6.2 RQ1 Results: Minimize nesting rule

This section discusses the accuracy and time results for rule R1: minimize nesting. Eye tracking results for tasks that pertain to rule R1 can be found in Section 6.4.

6.2.1 Comprehension Time

Comprehension time was measured by the time taken to read the snippet and time taken to answer the questions specified in Section 5.1. The timing information was provided by Tobii Studio software as it keeps timestamps for each task. As shown in Table 9, in the online study by Johnson et al. (2019), following the minimize nesting rule had a small but significant effect on reading time ($p < 0.001$, Cohen's $d = -0.481$) and a very small and insignificant effect on answer time ($p = 0.060$, Cohen's $d = -0.012$), while the eye tracking study as shown in Table 10 showed a medium but significant effect on reading time ($p < 0.001$, Cohen's $d = -0.757$), rejecting null hypothesis $H1_0$. Additionally, a very small and insignificant effect was observed on answer time ($p = 0.849$, Cohen's $d = -0.029$).

For the logical correctness of a snippet, the online study had a very small and insignificant effect on reading time of the snippet ($p = 0.119$, Cohen's $d = -0.097$) and a very small but significant effect on the answer time ($p = 0.005$, Cohen's $d = -0.175$). The eye tracking study displayed somewhat similar results with a very small and insignificant effect on reading time ($p = 0.618$, Cohen's $d = -0.075$) and a very small and insignificant effect on answer time ($p = 0.652$, Cohen's $d = -0.068$). A side-by-side comparison of the read and answer times in both studies is shown in Fig. 3.

Finding: Unlike the online study, the eye tracking study results show the logical correctness of the snippet had no significant effect on reading times. In both studies, snippets that followed the minimize nesting rule took less time to read.

6.2.2 Comprehension Confidence

The participants self-rated their confidence in the accuracy of their answer, with the average ranting of confidence for each task shown in Fig. 4. From this question, t-tests show from Table 9 that following the minimize nesting rule had a small but significant ($p < 0.001$, Cohen's $d = 0.452$) effect in the online study while the eye tracking study showed a medium and significant($p < 0.001$, Cohen's $d = 0.647$) effect, thereby rejecting the null hypothesis $H2_0$.

Table 9 Results for *t*-test for R1 (minimize nesting) and R2 (avoid do-while) from the online questionnaire study

		Readability	Comprehension Confidence	Comprehension Question	Logical Correctness	Read Time	Answer Time
R1 Logically	p	0.79348	0.18317	<0.00001	<0.00001	0.11926	0.00505
Correct	Cohen's d	-0.01630	-0.08292	0.39009	0.69869	-0.09707	-0.17493
R1 Rule	p	<0.00001	<0.00001	0.04953	0.25733	<0.00001	0.06023
Following	Cohen's d	1.09733	0.45175	0.12242	0.07056	-0.48068	-0.11712
R2 Logically	p	0.54257	0.02688	0.08911	<0.00001	0.23294	0.01953
Correct	Cohen's d	-0.03792	-0.13799	0.10595	0.55515	0.07431	-0.14562
R2 Rule	p	0.23558	0.85384	0.08911	0.78837	0.73196	0.90999
Following	Cohen's d	0.07389	0.01147	-0.10595	-0.01672	0.02133	-0.00704

Each dependent variable is shown per rule following and correctness criteria

Table 10 Results for the t-test for R1 and R2 from this eye tracking study

		Readability	Comprehension Confidence	Comprehension Question	Logical Correctness	Read Time	Answer Time
R1 Logically	p	0.52014	0.45125	<0.00001	0.01101	0.61781	0.65244
Correct	Cohen's d	0.09660	-0.11318	1.09039	0.38519	-0.07493	-0.06763
R1 Rule	p	<0.00001	0.00003	0.76164	0.34422	<0.00001	0.84892
Following	Cohen's d	1.02527	0.64696	-0.04554	0.14217	-0.75702	-0.02860
R2 Logically	p	0.37255	0.61933	0.54539	0.01619	0.36713	0.22844
Correct	Cohen's d	-0.13478	-0.07503	0.09134	0.36607	0.13632	-0.18221
R2 Rule	p	0.46582	0.83144	0.03324	0.17183	0.75696	0.96129
Following	Cohen's d	-0.11019	-0.03214	0.32355	0.20684	-0.04673	0.00733

Each dependent variable is shown per rule following and correctness criteria

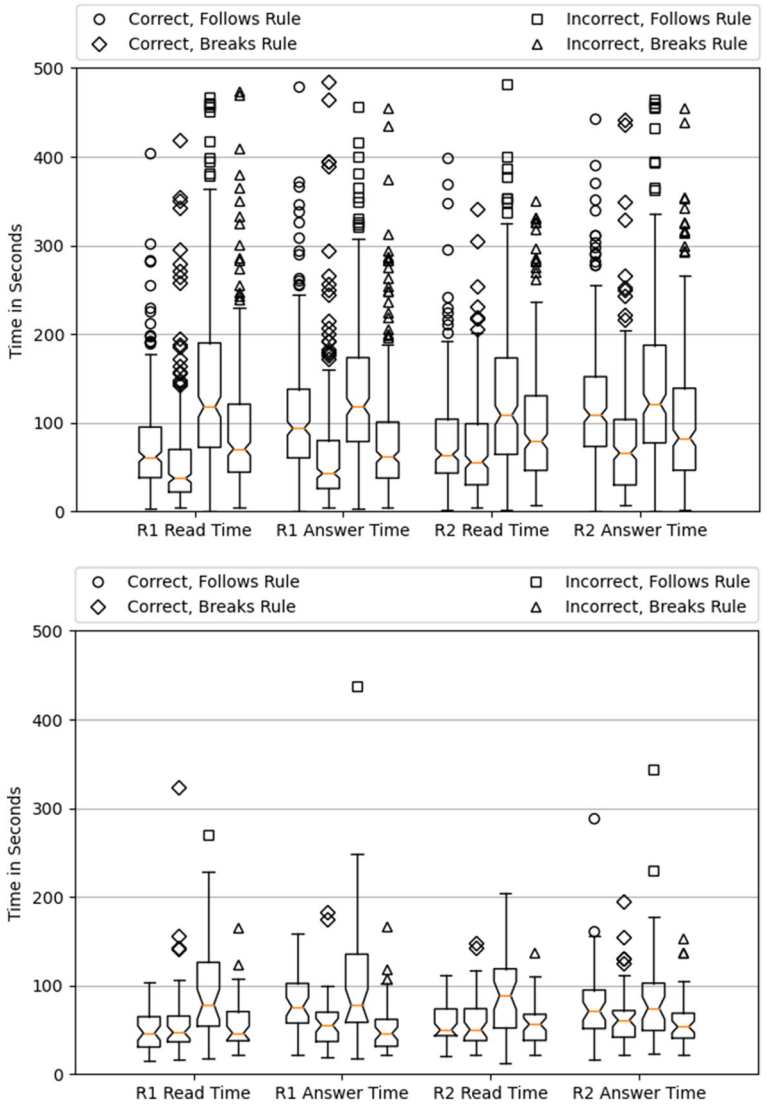

Fig. 3 Comparison of average completion time by snippet between studies. The top box plot is the online study from Johnson et al. (2019) while the bottom box plot is from the replication by this eye tracking study. Note that the incorrect, rule-following snippets consistently took the longest to both read and answer, while the correct, rule-breaking snippets consistently took the least time for the top plot, while rule-breaking snippets did not consistently take the least time for the bottom plot

Whether or not a snippet was logically correct had a very small and insignificant effect on comprehension confidence in both the online study ($p = 0.183$, Cohen's $d = -0.083$) and eye tracking study ($p = 0.451$, Cohen's $d = -0.113$).

Finding: Similar to the online study, following the minimize nesting rule in the eye tracking study made readers more confident in their understanding of a snippet. However, the effect is larger compared to the online study.

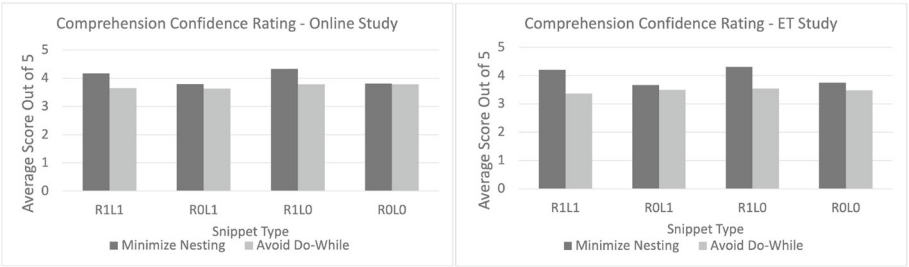

Fig. 4 Average Comprehension Confidence Rating by Snippet for each study. $P_k R_1 L_1$ snippets were both logically correct and followed the readability rule, $P_k R_0 L_1$ snippets were correct and broke the rule, $P_k R_1 L_0$ snippets were incorrect and followed the rule, and $P_k R_0 L_0$ snippets were incorrect and broke the rule

6.2.3 Level of Understanding

Two multiple-choice questions measured the level of understanding by a developer: a comprehension question, and a question asking whether or not the snippet was logically correct. Refer to Tables 9 and 10 for the t-test results for the online study (shown for reference) and the eye tracking study presented in this paper respectively. Refer to columns *Comprehension Question* and *Logical Correctness*.

In this eye tracking study, we show a very small and insignificant effect ($p = 0.762$, Cohen's $d = -0.046$) on whether or not the minimize nesting rule is followed and a large and significant ($p < 0.001$, Cohen's $d = 1.090$) effect for whether or not a snippet is logically correct, failing to reject the null hypothesis $H3_0$. On the other hand, in the online study it can be seen that following the minimize nesting rule has a very small but significant effect on how accurately a developer can answer multiple-choice questions about a given code snippet ($p = 0.050$, Cohen's $d = 0.122$) and a small but significant ($p < 0.001$, Cohen's $d = 0.390$) effect on comprehension question accuracy when the snippet is logically correct.

For the question of whether a snippet is logically correct, for the online study, following the minimize nesting rule had a very small and insignificant effect on the question correctness ($p = 0.257$, Cohen's $d = 0.071$) and whether or not a snippet was logically correct had a medium and significant effect ($p < 0.001$, Cohen's $d = 0.699$). However, the eye tracking study shows a small and insignificant effect ($p = 0.344$, Cohen's $d = 0.142$) for following the minimize nesting rule and a small but significant effect ($p = 0.011$, Cohen's $d = 0.385$) for its logical correctness.

Finding: Similar to the online study, whether a snippet was logically correct had a larger effect than whether the minimize nesting rule was followed on a participant's accuracy of comprehending a snippet and a participant's accuracy of determining the snippet's logical correctness.

6.2.4 Readability Assessment

Readability was assessed by a score, ranging from 0 to 5, given by each participant for each code snippet given to them. The average scores across both studies are shown in Fig. 5. In the online study as shown in Table 9, following the minimize nesting rule had a large and significant effect ($p < 0.001$, Cohen's $d = 1.097$) on the readability ratings, which is also in line with the eye tracking study results showing a large and significant ($p < 0.001$, Cohen's $d = 1.025$) effect in readability ratings.

On the other hand, the logical correctness of a snippet only had a very small and insignificant effect ($p = 0.794$, Cohen's $d = -0.016$) on the readability ratings for the online study, which is in line with the eye tracking study also showing a very small and insignificant ($p = 0.520$, Cohen's $d = 0.097$) effect on readability ratings.

Finding: The eye tracking study displays similar results to the online study, where following the minimize nesting rule made readers perceive code as more readable but its logical correctness had no significant effect on perceived readability.

6.2.5 Readability Rating - Method Comparison Task

These results are related to the method comparison question where two snippets were shown side by side (left - follows the rule, and right - does not follow the rule). The eye tracking results are presented in Section 6.4.2.

Out of 46 participants in the eye tracking study, 37 participants (80.43%) preferred the snippet that follows the readability rule R1: minimize nesting. The eye tracking study results align with the online study for R1, where 86.82% of participants responded with the rule-following snippet to have higher readability than the snippet that does not follow the rule.

Developers expressed strong opinions regarding the preference for following R1. For example, some comments from participants on R1 were:

- "*The one on the right seems obfuscated more than anything. Extra conditions don't help readability.*"
- "*Nested if statements are very difficult to read and follow. The left solution does not contain any nested if statements, while the right contains many nested if statements.*"

Finding: Participants in both studies rated the minimize nesting rule to be important.

6.3 RQ2 Results: Avoid do-while rule

This section discusses the accuracy and time results for rule R2: avoid do-while in relation to our prior study. Eye tracking results for tasks that pertain to rule R1 can be found in Section 6.5.

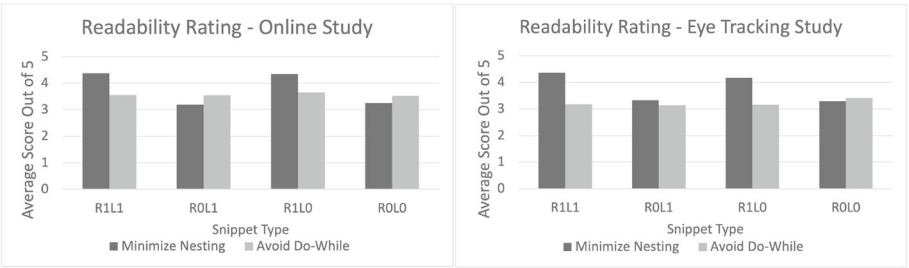

Fig. 5 Average Readability Rating by Snippet for each study. $P_k R_1 L_1$ snippets were both logically correct and followed the readability rule, $P_k R_0 L_1$ snippets were correct and broke the rule, $P_k R_1 L_0$ snippets were incorrect and followed the rule, and $P_k R_0 L_0$ snippets were incorrect and broke the rule

6.3.1 Comprehension Time

As with the minimize nesting rule, the comprehension time for the avoid do-while rule was measured by the time to read the snippet and the time to answer the question. As shown in Table 9, following the avoid do-while rule in the online study had a very small and insignificant effect on both the reading time and answer time (reading time: $p = 0.732$, Cohen's $d = 0.021$, answer time: $p = 0.910$, Cohen's $d = -0.007$). The eye tracking study also had a very small and insignificant effect on both the reading time ($p = 0.757$, Cohen's $d = -0.047$) and answer time ($p = 0.961$, Cohen's $d = 0.007$) as shown in Table 10 and therefore failing to reject null hypothesis $H5_0$.

For the logical correctness of a snippet, the online study had a very small and insignificant effect on the reading time of the snippet for the reading time ($p = 0.233$, Cohen's $d = 0.074$), and a very small but significant effect on the answer time ($p = 0.020$, Cohen's $d = -0.146$). The eye tracking study displayed a very small and insignificant effect for both reading ($p = 0.367$, Cohen's $d = 0.136$) and answer times ($p = 0.228$, Cohen's $d = -0.182$).

Finding: Unlike the online study, the eye tracking study results show no significant effects in comprehension time on snippets evaluating the avoid do-while rule regardless of whether the snippet was logically correct or following the rule.

6.3.2 Comprehension Confidence

Similar to RQ1, the participants were asked to self-rate their confidence in their response to the comprehension question with the average ranting of confidence for each task shown in Fig. 4. From this question, t-tests show from Tables 9 and 10 that following the avoid do-while rule had a very small and insignificant effect in both the online study and this eye tracking study (online study: $p = 0.854$, Cohen's $d = 0.012$, eye tracking study: $p = 0.831$, Cohen's $d = -0.032$), failing to reject the null hypothesis $H6_0$.

Whether or not a snippet was logically correct had a very small but significant effect on comprehension confidence in the online study ($p = 0.027$, Cohen's $d = -0.138$) and a very small and insignificant effect on the eye tracking study ($p = 0.619$, Cohen's $d = -0.075$).

Finding: Both studies show no significance on the confidence of a participant's accuracy of their responses for snippets evaluating the avoid do-while rule.

6.3.3 Level of Understanding

Similar to the minimize nesting rule, the avoid do-while rule also used two multiple-choice questions to measure the level of understanding by a developer: a comprehension question and a question asking whether or not a given snippet was logically correct. For these two questions, the t-tests results are also shown in Table 9. Similar to the results for rule R1: minimize nesting, the online study showed whether or not a snippet was logically correct had a larger impact in question accuracy than whether or not the avoid do-while readability rule was followed.

In the comprehension question, the online study had a very small and insignificant ($p = 0.089$, Cohen's $d = -0.106$) effect for following the avoid do-while rule and a small but insignificant effect ($p = 0.089$, Cohen's $d = 0.106$) for whether or not it is logically correct. The effects are also small in the eye tracking study, with a small but significant ($p = 0.033$, Cohen's $d = 0.324$) effect for following the avoid do-while rule and a very small and insignificant ($p = 0.545$, Cohen's $d = 0.091$) effect for its logical correctness, rejecting the null hypothesis $H7_0$.

For the question asking for logical correctness, the online study had a very small and insignificant ($p = 0.788$, Cohen's $d = -0.017$) effect for following the avoid do-while rule and a medium and significant ($p < 0.001$, Cohen's $d = 0.555$) effect for its logical correctness. The effects are again lesser in the eye tracking study, with a small and insignificant ($p = 0.172$, Cohen's $d = 0.207$) effect for following the rule and a small but significant ($p = 0.016$, Cohen's $d = 0.366$) effect for its logical correctness.

Finding: Unlike the online study, a snippet following the avoid do-while rule has a larger effect on a participant's accuracy in comprehending a snippet. The logical correctness of a snippet has a larger effect on a participant's accuracy in determining whether the snippet is logically correct.

6.3.4 Readability Assessment

Readability for the avoid do-while rule was also assessed by the readability rating given by each participant for each code snippet as shown in Fig. 5. In the online study as shown in Table 9, following the avoid do-while rule had a very small and insignificant effect ($p = 0.236$, Cohen's $d = 0.0739$) on the readability ratings, which is also in line with the eye tracking study results also showing a very small and insignificant ($p = 0.466$, Cohen's $d = -0.110$) effect in readability ratings as shown in Table 10.

The logical correctness of a snippet had a very small and insignificant effect on the readability ratings for both studies (online study: $p = 0.543$, Cohen's $d = -0.038$, eye tracking study: $p = 0.373$, Cohen's $d = -0.135$) effect on readability ratings.

Finding: Both studies show that following the avoid do-while rule, the snippet's logical correctness has no significant effect on perceived readability.

6.3.5 Readability Rating - Method Comparison Task

These results are related to the method comparison question where two snippets were shown side by side (left - follows the rule, and right - does not follow the rule). The eye tracking results are presented in Section 6.5.2.

Out of 44 participants, 25 participants in the eye tracking study (56.82%) preferred the snippet that follows the readability rule R2: avoid do-while loops. Similar to the results of RQ1, the results for RQ2 also corroborate with the online study, where 67.44% of the study participants rated snippets following R2 as having higher readability.

Developers did not have as strong of an opinion regarding the preference for following R2. While the quoted examples in rule R1 appear to display outright frustration, this does not appear to be the case for R2. For example, the same participants quoted from RQ1 stated regarding R2:

- *"Having the condition before the loop makes it easier to digest the context of the loop code."*
- *"The left has less lines of code to read. The empty string check is built in and the while loop check is easier to read and understand with the left code. The first thing I look for in this problem is whether or not the bounds are correct, so having the while loop check in the beginning makes that easier to see and remember."*

Finding: Similar to the online study, participants rated the avoid do-while rule as important but not as important as the minimize nesting rule.

6.4 RQ3 Results: Eye Movement Behavior for Minimize Nesting Rule

This section presents results for the eye tracking metrics for the minimize nesting rule for the method analysis and method comparison tasks. All metrics are normalized to account for the length/token difference in snippet variants: The gaze linearity metrics are a percentage of occurrences across all fixations, fixation counts are the number of fixations per second by dividing for task duration in seconds, and fixation durations are divided by the number of tokens in each snippet. For the method analysis tasks, we present two types of comparisons (general and pairwise). The general comparisons compare a) rule following regardless of correctness and b) logical correctness regardless of rule-following. The pairwise comparisons compare rule-following and logically correct with three other combinations namely, a) rule-breaking and logically correct, b) rule-following and logically incorrect and c) rule-breaking and logically incorrect.

6.4.1 Method Analysis Tasks

Descriptive Statistics

When looking at single method analysis tasks as defined in Section 5.1, the fixation counts and fixation durations after normalizing for task duration and snippet code length can be seen in Fig. 6. Going from a rule-following to rule-breaking snippet with both snippets being logically correct ($P_k R_1 L_1$ vs. $P_k R_0 L_1$) did not show a noticable difference in the average number of fixations and fixation durations on the code snippet. In addition, going from a logically correct snippet to a logically incorrect snippet appear to have similar fixation counts and durations as shown in Fig. 6 ($P_k R_1 L_1$ vs. $P_k R_1 L_0$, $P_k R_0 L_1$ vs. $P_k R_0 L_0$).

Next, we present the eye tracking metrics derived by Busjahn et al. (2015). The distributions for the rate of the overall vertical next text, vertical later text, and line coverage vary more than the other three metrics as shown in Fig. 7 with standard deviations of 15.99%, 17.81%, and 6.02% respectively. For all metrics except line coverage, the percentages for each of the metrics are lower when a snippet does not follow rule R1 regardless of its logical correctness. The differences are more pronounced when the snippets are logically correct and for vertical next text, vertical later text, logically correct snippets have higher averages while for horizontal later text, regression rate, and line regression rate the logically incorrect snippets have higher averages.

General Comparisons - Minimize Nesting

To see whether these differences had any statistical significance, we performed the paired Wilcoxon signed-rank test on each of these metrics, as the data is not normally distributed for most metrics. Cohen's d was used to determine the effect size. Table 11 shows our results to determine whether there is a significant effect on the eye tracking metrics in regards to whether or not the readability rule R1 was followed ($R_1 L_1$, $R_1 L_0$ vs. $R_0 L_1$, $R_0 L_0$), or the snippet was logically correct ($R_1 L_1$, $R_0 L_1$ vs. $R_1 L_0$, $R_0 L_0$). We also corrected for each metric using the Benjamini-Hochberg False Discovery Rate (FDR) procedure (Benjamini and Hochberg 1995) with a base $\alpha = 0.05$. In other words, the lower p-value for each row is significant if it is lower than $\frac{1}{2} \times 0.05$ and the higher p-value is significant if it is lower than 0.05.

As shown in Table 11, there was a very small significant effect observed on the **fixation counts** ($p < 0.001$, Cohen's $d = 0.1113$) when comparing a rule-following snippet against a snippet that did not follow rule R1 with the rule breaking snippets having lower counts. There was also a very small effect on fixation counts when the snippet was logically incorrect

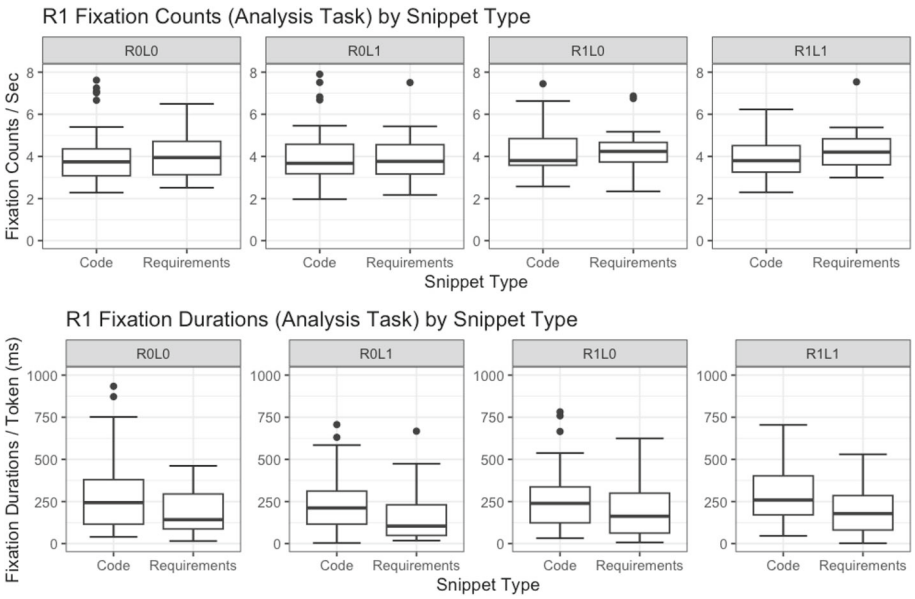

Fig. 6 Fixation counts (top) and durations (bottom) for single method analysis code snippets for rule R1: minimize nesting. Each box represents the problem statement (Requirements) and code for each snippet type. ($P_k R_1 L_1$ - snippets that were logically correct and followed the readability rule, $P_k R_0 L_1$ - logically correct and did not follow the rule, $P_k R_1 L_0$ - incorrect and followed rule, and $P_k R_0 L_0$ - incorrect and broke the rule)

($p = 0.008$, Cohen's $d = -0.1084$), but the logically incorrect snippets have lower fixation counts. For the gaze-based measures of linearity, there was a lower rate of **horizontal later text**, or the likelihood of a gaze moving forward within a line, in snippets that were rule-

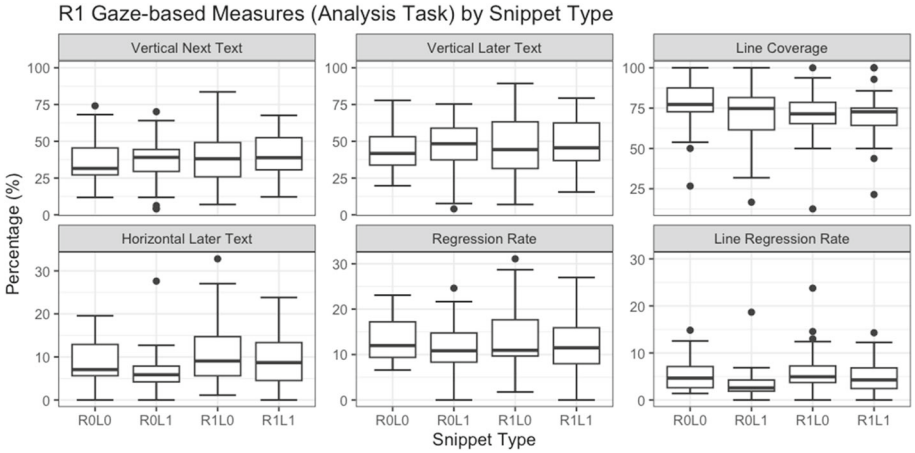

Fig. 7 Distribution of gaze-based measures for single method analysis code snippets for rule R1: minimize nesting. Each box represents the individual measures, separated by $P_k R_1 L_1$ - snippets that were logically correct and followed the readability rule, $P_k R_0 L_1$ - logically correct and did not follow the rule, $P_k R_1 L_0$ - incorrect and followed rule, and $P_k R_0 L_0$ - incorrect and broke the rule

Table 11 Statistical tests of gaze-based metrics and fixation counts and durations of *single method analysis* tasks testing rule R1: minimize nesting

	Rule-Breaking		Logically Incorrect		Shapiro-Wilk
Metric Name	***p***	**Cohen's *d***	***p***	**Cohen's *d***	***W* (*p*)**
Vertical next text	0.1997	0.1615	0.7755	-0.0362	0.990 (0.514)
Vertical later text	0.2651	0.1389	0.4901	0.0883	0.990 (0.583)
Horizontal later text	*0.0094	0.3296	*0.0054	-0.3023	0.912 (<0.001)
Regression rate	0.4413	0.1204	*0.0057	-0.3439	*0.970 (0.009)
Line regression rate	*0.0106	0.2985	*0.0002	-0.4362	*0.862 (<0.001)
Line coverage	0.0627	-0.1222	0.1330	-0.1525	*0.914 (<0.001)
Saccade length	0.1032	0.1760	0.3626	-0.0976	0.984 (0.181)
Fixation count	*<0.0001	0.1113	*0.0084	-0.1084	*0.903 (<0.001)
Fixation duration	0.0413	0.1120	0.4594	-0.0460	*0.868 (<0.001)

The tests are corrected for multiple comparison on a per-row basis using the False Discovery Rate procedure Significant comparisons are marked with an asterisk * before the p-value

breaking with a small effect ($p = 0.009$, Cohen's $d = 0.330$). There was a higher rate of horizontal later text in snippets that were logically incorrect with a small effect ($p = 0.005$, Cohen's $d = -0.302$). A lower rate of **line regression rate**, or the rate of gazes moving backwards within a line, was observed for snippets with a small effect that were rule-breaking ($p = 0.011$, Cohen's $d = 0.299$). A higher rate of line regresison rate was observed for snippets that were logically incorrect with a small effect ($p < 0.001$, Cohen's $d = -0.436$). There was also a higher **regression rate**, or the rate of a gaze moving backwards, when a snippet is logically incorrect with a small effect ($p = 0.006$, Cohen's $d = -0.344$).

Out of the variables marked with an asterisk in Table 11, we can reject the null hypothesis $H4_0$ for horizontal later text, line regression rate, and fixation counts when a snippet follows rule R1.

Finding: A snippet following rule R1 results in higher fixation counts, higher rates of horizontal later text and line regression. Similarly, a logically correct snippet results in a higher fixation count and higher rate of horizontal later text, regression rate, and line regression rate.

Pairwise Comparisons - Minimize Nesting

In this section, we compare the treatment where a given snippet is both rule-following and logically correct against a treatment that is rule-breaking but logically correct (R_1L_1 vs. R_0L_1), a treatment that is rule-following but logically incorrect (R_1L_1 vs. R_1L_0), and a treatment that is rule-breaking and logically incorrect (R_1L_1 vs. R_0L_0). This is in contrast to the previous section where in addition to comparing two treatments that are logically correct against two treatments that are logically incorrect (R_1L_1, R_1L_0 vs. R_0L_1, R_0L_0) and two rule-following treatments against two rule-breaking treatments (R_1L_1, R_0L_1 vs. R_1L_0, R_0L_0). The results for these comparisons can be seen in Table 12. We also correct for multiple comparisons using the FDR procedure. As there are 3 comparisons made for each metric, the lowest p-value for each row is significant if it is lower than $\frac{1}{3} \times 0.05$, the second lowest p-value for each row is significant if it is lower than $\frac{1}{2} \times 0.05$, and the highest p-value is significant if it is lower than 0.05.

Table 12 shows that out of the eye tracking metrics, there was a very small and significant effect on fewer **fixation counts** ($p = 0.007$, Cohen's $d = 0.133$) and a small effect on shorter

fixation durations ($p = 0.010$, Cohen's $d = 0.2616$) when the snippet was violating rule R1. However, the fixation counts and fixation durations were not statistically significant when the snippet was only logically incorrect or when the snippet was both rule-breaking and logically incorrect.

Finding: When doing a pairwise comparison for single method analysis tasks, snippets that did not follow the minimize nesting rule R1 had a lower rate of fixation counts and fixation durations.

6.4.2 Method Comparison Tasks

For the method comparison tasks, we present descriptive statistics of the side-by-side comparisons followed by the statistical results on the eye tracking metrics. Please refer to Fig. 8 for an example of fixations of a participant performing the side-by-side comparison task.

Descriptive Statistics

Participants had a higher number of fixations with longer fixation durations on code that does not follow rule R1: minimize nesting compared to code that does follow the rule. Figure 9 shows the distribution of the number of fixations and the duration of fixations. The problem statement has an average of 27.84 fixations and an average fixation durations of 6.74 seconds. Code snippets that follow rule R1 (snippet $P_k R_1 L_1$) had an average of 42.82 fixations with 10.7 seconds of average fixation durations compared to the snippet that did not follow rule R1 (snippet $P_k R_0 L_1$, 47.65 average fixations with 12.03 seconds average fixation duration).

Out of the gaze-based measures specified by Busjahn et al. (2015), the rate of the overall vertical next text, vertical later text, and line coverage have wider distributions than the other three metrics, with standard deviations of 12.69%, 14.16%, and 19.83% respectively. The difference in these measures by whether or not a given snippet is following rule R1 can be seen in Fig. 10.

Table 12 Pairwise statistical tests of gaze-based metrics and fixation counts and durations of *single method analysis* tasks testing rule R1: minimize nesting

	Rule-Breaking		Logically Incorrect		Both	
Metric Name	***p***	**Cohen's *d***	***p***	**Cohen's *d***	***p***	**Cohen's *d***
Vertical next text	0.5221	0.2276	0.9491	0.0123	0.5221	0.1379
Vertical later text	0.6089	0.1640	0.8314	0.1003	0.2842	0.2622
Horizontal later text	0.0291	0.4721	0.4420	-0.1812	0.8842	0.0127
Regression rate	0.5940	0.1411	0.4812	-0.2720	0.3052	-0.2904
Line regression rate	0.0655	0.4275	0.4549	-0.3039	0.6136	-0.1956
Line coverage	0.8882	0.0331	0.4611	0.0088	0.0880	-0.2967
Saccade length	0.4420	0.1114	0.3692	-0.1614	0.7114	0.1095
Fixation counts	*0.0072	0.1331	0.2705	-0.0914	0.6779	0.0016
Fixation durations	*0.0100	0.2845	0.2616	0.1209	0.7580	0.0650

The tests are corrected for multiple comparison on a per-row basis using the False Discovery Rate procedure. Significant comparisons are marked with an asterisk * before the p-value

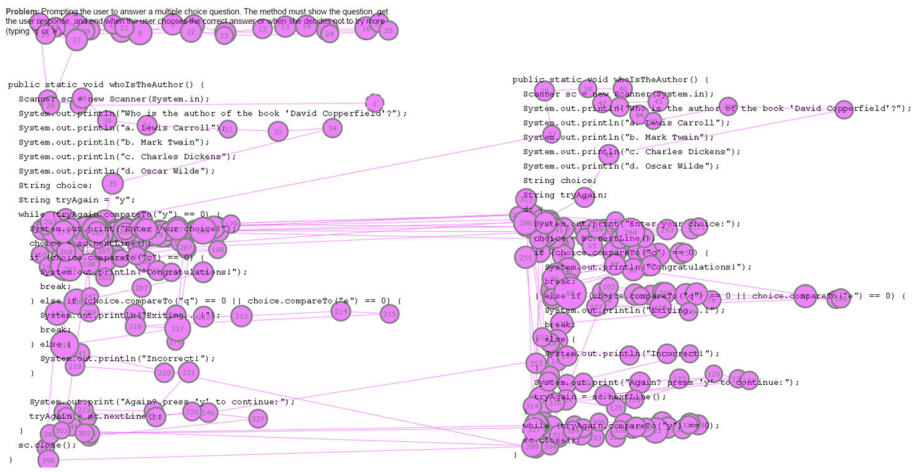

Fig. 8 Fixation graph from one participant doing the method comparison task for R2: avoid do-while. Circles represent fixations with radius indicating the duration of the fixation. The method functionality is given on top

Most gaze-based measures for method comparison tasks regarding rule R1 display similar averages. The notable exception to this is the line coverage that is significantly higher when the minimize nesting rule is not followed (21.56% following rule R1 vs. 34.83% not following rule R1). However, when looking at the distribution of the measures as shown in Fig. 10, the distributions for line coverages are visibly wider for the non rule-following snippets vs. the rule following snippets with a standard deviation nearly twice as large (13.3% vs. 6.93% respectively). A large difference in the range of distributions can also be seen with the rate of horizontal later text, where the rule following snippet has a wider distribution vs. the non-rule following snippet with a standard deviation of 12.31% vs. 6.57%, respectively.

Statistical Analysis - Minimize Nesting To determine whether there was statistical significance in the eye tracking data, we used the paired Wilcoxon signed-rank test. As shown in

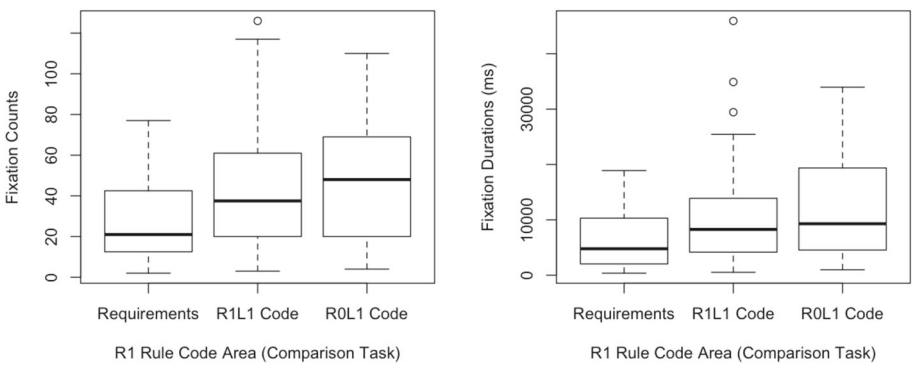

Fig. 9 Fixation counts and durations for side-by-side method comparison code snippets for rule R1: minimize nesting. Each box represents the problem statement (Requirements), Following R1 ($P_k R_1 L_1$ Code), and not Following R1 ($P_k R_0 L_1$ Code)

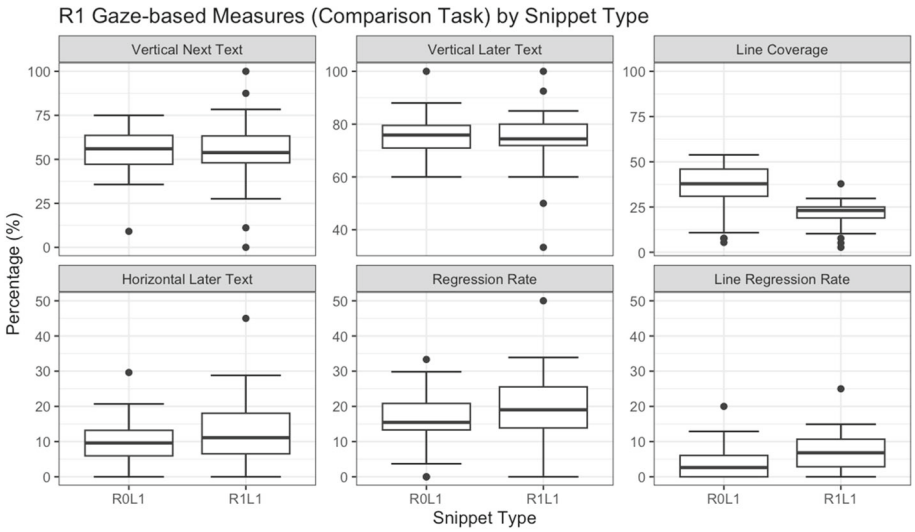

Fig. 10 Distribution of gaze-based measures for side-by-side method comparison code snippets for rule R1: minimize nesting. Each box represents the individual measures, separated by $P_k R_1 L_1$ - snippets that were logically correct and followed the readability rule and $P_k R_0 L_1$ - logically correct and did not follow the rule

Table 13, there is a very small to small effect for most metrics except for the line regression rate and the line coverage, but the effects are insignificant. For the statistically significant effects of eye movement behavior, we reject the null hypothesis $H4_0$ for the metrics of line regression rate and line coverage. There is an decrease on the **line regression rate** with a medium effect ($p = 0.001$, Cohen's $d = 0.578$) when the snippet is rule-breaking. There is an increase on the **line coverage** with a very large effect ($p < 0.001$, Cohen's $d = -1.231$) when the snippet is rule-breaking.

Finding: There is a lower rate of line regression and higher rate of line coverage on rule-breaking snippets in side-by-side method comparison tasks evaluating rule R1.

Table 13 Statistical tests of gaze-based metrics and fixation counts and durations of *side-by-side method comparison* tasks testing rule R1: minimize nesting

Metric Name	*p*	Cohen's *d*	Effect Size	Shapiro-Wilk
Vertical next text	0.7169	-0.0473	Very Small	*0.928 (0.009)
Vertical later text	0.7929	-0.1309	Very Small	*0.932 (0.012)
Horizontal later text	0.0696	0.4245	Small	0.975 (0.452)
Regression rate	0.0737	0.3816	Small	0.977 (0.522)
Line regression rate	*0.0012	0.5780	Medium	*0.948 (0.045)
Line coverage	*<0.0001	-1.2312	Very Large	*0.875 (<0.001)
Saccade length	0.9857	0.1018	Very Small	*0.945 (0.035)
Fixation counts	0.5826	-0.1307	Very Small	*0.927 (<0.001)
Fixation durations	0.7514	-0.1178	Very Small	*0.867 (<0.001)

Significant comparisons are marked with an asterisk * before the p-value

6.5 RQ4 Results: Eye Movement Behavior for Avoid do-while Rule

This section presents the eye tracking metrics results for the avoid do-while rule for the method analysis and method comparison tasks. We perform similar general and pairwise comparisons in this section as done for RQ3.

6.5.1 Method Analysis Tasks

Descriptive Statistics For the single method analysis tasks as defined in Section 5.1, participants had a lower number of fixations with shorter fixation durations on code that does not follow rule R2: avoid do-while compared to code that does follow the rule as shown in Fig. 11. Going from a rule-following to a rule breaking snippet with both snippets being logically correct resulted in a decrease of both the average number of fixations and fixation durations as shown in the figure ($P_k R_1 L_1$ vs. $P_k R_0 L_1$). However, whether the code snippet follows rule R2 does not appear to make a difference when both snippets are logically incorrect ($P_k R_1 L_0$ vs. $P_k R_0 L_0$).

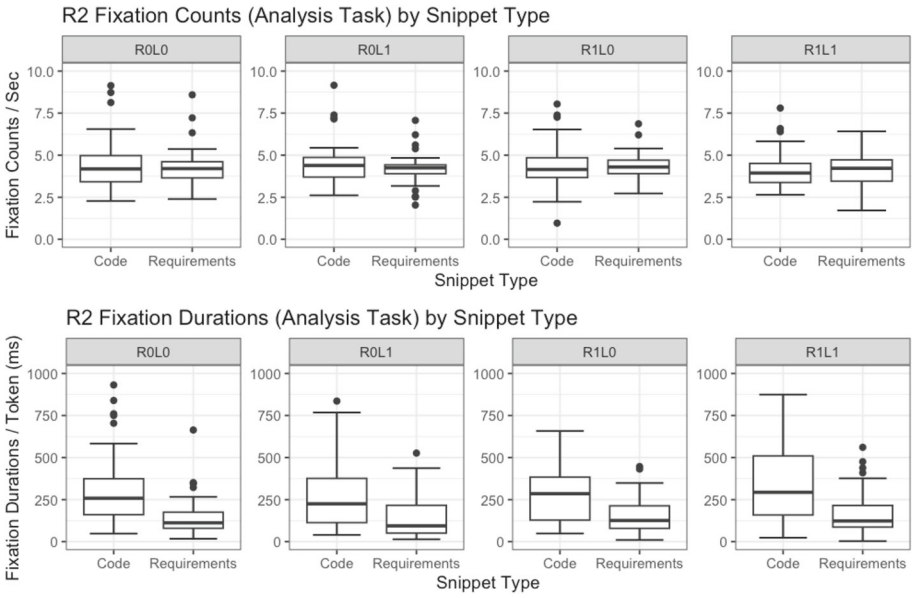

Fig. 11 Fixation counts (top) and durations (bottom) for *single method analysis* code snippets for rule R2: avoid do-while loops. Each box represents the problem statement (Requirements) and code for each snippet type. ($P_k R_1 L_1$ - snippets that were logically correct and followed the readability rule, $P_k R_0 L_1$ - logically correct and did not follow the rule, $P_k R_1 L_0$ - incorrect and followed rule, and $P_k R_0 L_0$ - incorrect and broke the rule)

The distributions for the rate of the overall vertical next text, vertical later text, and line coverage vary more than the other three metrics as shown in Fig. 12 which is further corroborated by the standard deviations of 15.73%, 16.9%, and 6.01% respectively.

Figure 12 also shows that for all metrics except vertical later text, the percentage for each metric is lower when a snippet does not follow rule R2 regardless of its logical correctness. The differences are more pronounced when the snippets are logically correct and for horizontal later text, logically correct snippets have higher averages while for vertical next text, vertical later text, and regression rate the logically incorrect snippets have higher averages.

General Comparisons - Avoid do-while Similar to RQ3, we performed the paired Wilcoxon signed-rank test to determine statistical significance due to the data not being normally distributed with Cohen's d used to determine the effect size. Table 14 shows our results to determine whether there is a significant effect on the eye tracking metrics in regards to whether or not the readability rule R2 was followed (R_1L_1, R_1L_0 vs. R_0L_1, R_0L_0), or the snippet was logically correct (R_1L_1, R_0L_1 vs. R_1L_0, R_0L_0). As we did with RQ3, we also applied FDR correction for each metric with a base $\alpha = 0.05$. In other words, the lower p-value for each row is significant if it is lower than $\frac{1}{2} \times 0.05$ and the higher p-value is significant if it is lower than 0.05.

As shown in Table 14, there was an increase on the **fixation counts** with a very small effect ($p = 0.001$, Cohen's $d = -0.151$) when a snippet was not following rule R2. However, a decrease in **fixation durations** with a very small effect was observed ($p = 0.004$, Cohen's $d = 0.188$). There was higher number of **fixation counts** observed with a very small effect ($p = 0.03$, Cohen's $d = -0.101$) when a snippet was not logically correct. Unlike our comparisons on snippets evaluating rule R1, there were no significant effects on the gaze-based linearity metrics observed.

We reject the null hypothesis $H8_0$ for the two metrics marked with an asterisk, fixation counts and fixation duration.

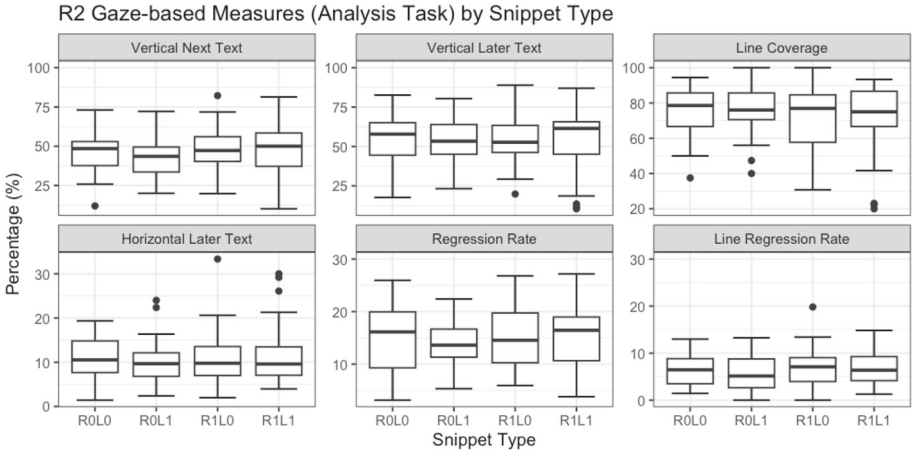

Fig. 12 Gaze-based measures for single method analysis code snippets for rule R2: avoid do-while. Each box represents the individual measures, separated by $P_kR_1L_1$ - snippets that were logically correct and followed the readability rule, $P_kR_0L_1$ - logically correct and did not follow the rule, $P_kR_1L_0$ - incorrect and followed rule, and $P_kR_0L_0$ - incorrect and broke the rule

Table 14 Statistical tests of gaze-based metrics and fixation counts and durations of *single method analysis* tasks testing rule R2: avoid do-while

	Rule-Breaking		Logically Incorrect		Shapiro-Wilk
Metric Name	***p***	**Cohen's *d***	***p***	**Cohen's *d***	***W* (*p*)**
Vertical next text	0.1180	0.1461	0.2616	-0.1683	0.987 (0.320)
Vertical later text	0.6307	0.0177	0.8031	-0.0927	0.978 (0.056)
Horizontal later text	0.1280	0.1983	0.8611	0.0077	*0.923 (<0.001)
Regression rate	0.8828	0.0486	0.4429	-0.1475	0.986 (0.296)
Line regression rate	0.0781	0.1977	0.7755	-0.1020	*0.966 (0.004)
Line coverage	0.3230	-0.1525	0.8332	-0.0151	*0.923 (<0.001)
Saccade length	0.0422	-0.1511	0.9637	-0.0559	0.994 (0.912)
Fixation counts	*0.0013	-0.1508	*0.0297	-0.1007	*0.903 (<0.001)
Fixation durations	*0.0036	0.1876	0.6101	0.0413	*0.751 (<0.001)

The tests are corrected for multiple comparison on a per-row basis using the False Discovery Rate procedure Significant comparisons are marked with an asterisk * before the p-value

Finding: Snippets that do not follow the avoid do-while rule have higher fixation counts and shorter fixation durations and logically incorrect snippets have higher fixation counts. However, neither had any effect on how the code was navigated as measured by the linearity metrics.

Pairwise Comparisons - Avoid do-while

Similar to the additional comparisons we performed in Section 6.4.1, we compared the treatment where a given snippet is both rule-following and logically correct against a treatment that is rule-breaking but logically correct (R_1L_1 vs. R_0L_1), a treatment that is rule-following but logically incorrect (R_1L_1 vs. R_1L_0), and a treatment that is rule-breaking and logically incorrect (R_1L_1 vs. R_0L_0). The results for these comparisons can be seen in Table 15. We also correct for multiple comparisons using the FDR procedure. As there are 3 comparisons made for each metric, the lowest p-value for each row is significant if it is lower than $\frac{1}{3} \times 0.05$, the second lowest p-value for each row is significant if it is lower than $\frac{1}{2} \times 0.05$, and the highest p-value is significant if it is lower than 0.05.

The results in Table 15 show that there were no significant effects on the gaze-based measures of linearity when comparing a rule-following, logically correct snippet against either a rule-breaking and logically correct (R_1L_1 to R_0L_1), rule-following and logically incorrect (R_1L_1 to R_1L_0), or rule-breaking and logically incorrect snippet (R_1L_1 to R_0L_0). There was a small and significant effect on the fixation counts when the snippet was rule-breaking ($p = .002$, Cohen's $d = -0.2742$), logically incorrect ($p = 0.013$, Cohen's $d = -0.2150$), or both rule-breaking and logically incorrect ($p = 0.008$, Cohen's $d = -0.233$).

Finding: Pairwise comparisons show that whether a snippet followed rule R2 or was logically correct did not have an effect on any eye tracking metrics (except higher fixation counts when a snippet is either rule-breaking, logically incorrect, or both) for single method analysis tasks.

Table 15 Statistical tests of gaze-based metrics and fixation counts and durations of *single method analysis* tasks testing rule R2: avoid do-while

	Rule-Breaking		Logically Incorrect		Both	
Metric Name	***p***	**Cohen's *d***	***p***	**Cohen's *d***	***p***	**Cohen's *d***
Vertical next text	0.2470	0.1545	0.7655	-0.1291	0.6856	-0.0198
Vertical later text	0.4812	0.0335	0.8983	-0.0634	0.6856	-0.0668
Horizontal later text	0.1316	0.3238	0.4294	0.1259	0.4294	0.1980
Regression rate	0.6239	0.0763	0.8815	-0.1152	0.7493	-0.0909
Line regression rate	0.1757	0.2278	0.8148	-0.0694	0.5504	0.1013
Line coverage	0.5026	-0.2108	0.8968	-0.0711	0.6189	-0.1612
Saccade Length	0.4946	-0.1825	0.9491	-0.0830	0.1207	-0.1966
Fixation counts	*0.0023	-0.2742	*0.0128	-0.2150	*0.0077	-0.2330
Fixation durations	0.0347	0.2618	0.2969	0.1114	0.0876	0.2311

The tests are corrected for multiple comparison on a per-row basis using the False Discovery Rate procedure
Significant comparisons are marked with an asterisk * before the p-value

6.5.2 Method Comparison Tasks

Descriptive Statistics

Participants had a higher number of fixations with longer fixation durations on code that follows rule R2: avoid do-while compared to code that does not follow the rule. Figure 13 shows the distribution of the number of fixations and the duration of fixations. The problem statement has an average of 29.58 fixations and an average fixation durations of 7.74 seconds. Code snippets that follow R2 (snippet $P_k R_1 L_1$) had an average of 88.32 fixations with 21.46 seconds of average fixation durations compared to the snippet that did not follow R2 (71.75 average fixations with 16.14 second average fixation duration).

Similar to the gaze-based measures shown in the side-by-side method comparison tasks for rule R1, the rate of vertical next text, vertical later text, and line coverage for rule R2 have wider distributions than the other three metrics with standard deviations of 12.49%, 9.88%,

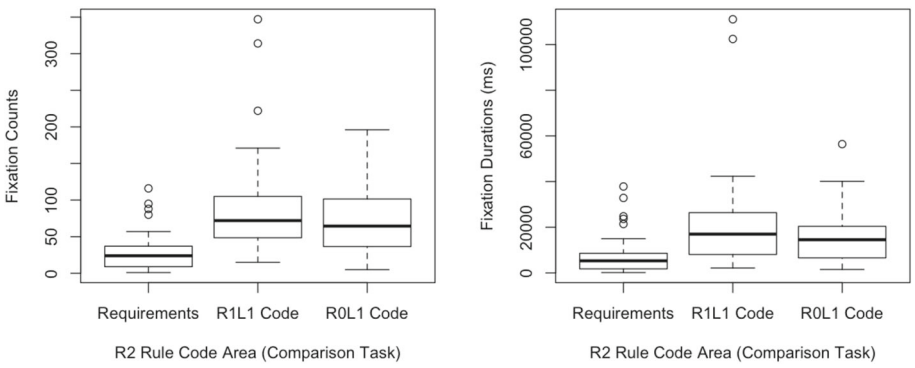

Fig. 13 Distribution of fixation counts and durations for *side-by-side* method comparison code snippets for rule R2: avoid do-while loops. Each box represents the problem statement (Requirements), Following R2 ($P_k R_1 L_1$), and not Following R2 ($P_k R_0 L_1$)

and 14% respectively. The differences in measures based on whether or not a given snippet follows rule R2 are shown in Fig. 14.

Compared to the gaze-based measures for side-by-side method comparison tasks of rule R1, the measures for rule R2 have higher averages for the rule following snippet with the exception of line coverage. The noticeable difference can be seen with the vertical next text for snippets that follow the avoid do-while rule vs. snippets that do not (60.03% vs. 51.94% respectively). This is also notable in Fig. 14, where the distributions are roughly similar for the rule following snippets vs. the non-rule following snippets with the exception of the line coverage metric. Therefore, the higher average can be noticed from the boxes representing $P_k R_1 L_1$ be above $P_k R_0 L_1$ except for the line coverage.

Statistical Analysis - Avoid do-while The paired Wilcoxon signed-rank test was used to determine whether there was statistical significance in any of our results for the side-by-side comparison task for rule R2. As shown in Table 16, there is a significant effect on all gaze-based linearity metrics, and we therefore reject the null hypothesis $H8_0$ for all of them.

Vertical next text results show a medium effect ($p = 0.001$, Cohen's $d = 0.578$), suggesting that gazes are less likely to move forward one line. A medium effect is observed with the **vertical later text** results ($p = 0.004$, Cohen's $d = 0.534$), suggesting that gazes are less likely to move forward multiple lines if the snippet is rule-breaking. **Horizontal later text** results show a small effect ($p = 0.004$, Cohen's $d = 0.331$) which shows that gazes are less likely to move forward within a line when reading a rule-breaking snippet. **Regression rate** results suggest gazes are less likely to move backwards in rule-breaking snippets with a medium effect ($p < 0.001$, Cohen's $d = 0.699$). **Line regression rate** results show a medium effect ($p < 0.001$, Cohen's $d = 0.680$) which suggests gazes are less likely to move

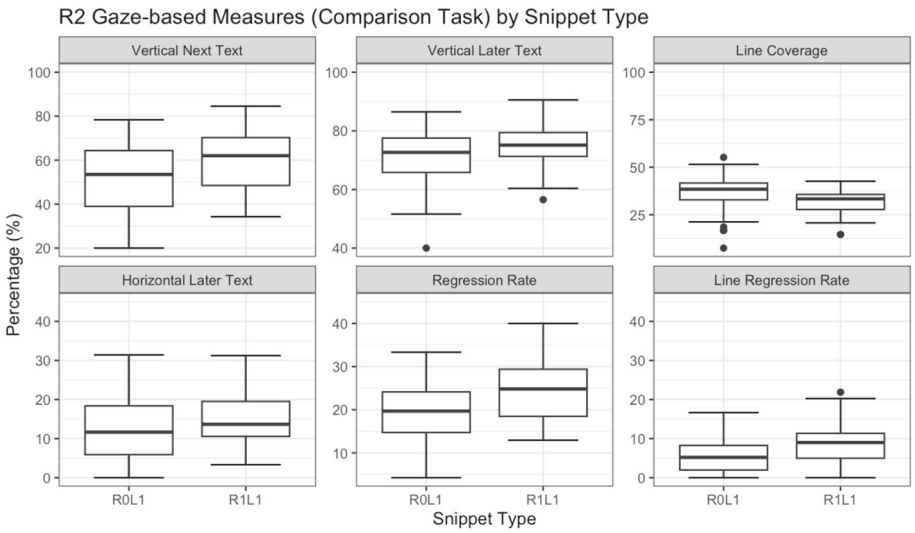

Fig. 14 Distribution of gaze-based measures for *side-by-side* method comparison code snippets for rule R2: avoid do-while. Each box represents the individual measures, separated by $P_k R_1 L_1$ - snippets that were logically correct and followed the readability rule and $P_k R_0 L_1$ - logically correct and did not follow the rule

Table 16 Statistical tests of gaze-based metrics and fixation counts and durations of *side-by-side method comparison* tasks testing rule R2: avoid do-while

Metric Name	*p*	Cohen's *d*	Effect Size	Shapiro-Wilk
Vertical next text	*0.0006	0.5777	Medium	0.954 (0.078)
Vertical later text	*0.0039	0.5335	Medium	*0.928 (0.009)
Horizontal later text	*0.0038	0.3315	Small	*0.935 (0.016)
Regression rate	*<0.0001	0.6993	Medium	0.972 (0.341)
Line regression rate	*<0.0001	0.6801	Medium	0.966 (0.216)
Line coverage	*0.0008	-0.5948	Medium	*0.909 (0.002)
Saccade length	0.4839	-0.0818	Very Small	*0.942 (0.027)
Fixation counts	0.0704	0.2829	Small	*0.822 (<0.001)
Fixation durations	0.0657	0.3041	Small	*0.719 (<0.001)

Significant comparisons are marked with an asterisk * before the p-value
The Shapiro-Wilk test shows the W statistic with the p-value in parentheses

backwards horizontally on a rule-breaking snippet. A medium effect is also observed for **line coverage** results ($p = 0.001$, Cohen's $d = -0.595$) that show there are gazes on a larger percentage of total lines on a rule-breaking snippet.

Finding: There is a small to medium effect on all gaze-based linearity metrics when participants are reading rule-following and rule-breaking snippets in side-by-side method comparison tasks evaluating rule R2. All statistically significant metrics were lower on the rule-breaking snippets except for line coverage which was higher.

6.6 RQ5 Results: Eye Movement Behavior vs. Preference Rating

This section presents the results of the preference rating of two pairs of correct code snippets shown side by side. First, we present results on the ranking of the code snippets, followed by how the eye movement behavior correlated with the rating for each rule. The goal was to determine if there was any correlation between the rating preference chosen and how participants read each of the code snippets when shown side by side.

6.6.1 Readability Rule Ranking Results

Following the method comparison task, participants were shown two methods side-by-side, with one method that followed a readability rule while the other that did not. All 46 participants answered questions regarding both rules R1: minimize nesting and R2: avoid do-while. Out of the 46 participants, 87% ranked snippets that follow rule R1: minimize nesting rule, to be more readable than snippets that do not, and 56.5% ranked snippets following rule R2: avoid do-while as more important than snippets that do not follow the rule. These results are similar to the results from the online study, where 86.82% of participants ranked snippets following rule R1 higher and 67.44% of participants ranked snippets following rule R2 higher.

For the minimize nesting snippets, 6 participants rated the rule-breaking snippet to be more readable. Out of the 6, only one participant provided an explanation for their choice. They stated and we quote verbatim:

"I found the right one more readable because the variable that was returned at the end instead of returning the value through all of the if statements. It made me check to see if it was greater than 25 if all of the if statements would run."

Of the 40 participants that rated the rule-following snippet higher, 30 participants gave explanations which suggested that the code was shorter and easier to follow. One of the explanations include the following quote:

"The right version of the code had so many nested if statements that it made it difficult to tell where control statements started and ended, and it was far larger then the left sample."

For the avoid do-while snippets, out of the 20 participants that rated the rule-breaking snippet higher, 13 participants provided explanations, where 6 explained the rule-breaking snippet assigned variables sooner, 5 simply stated the snippet was easier to read, and 2 stated it was due to preference. Out of the 26 participants that rated the rule-following snippet higher, 15 participants provided explanations, with 6 stating the code is shorter, 6 stated the do-while loops are less readable in general, and 3 simply stated they result in unnecessarily longer code.

Finding: Overall, participants ranked the snippets following the readability rules to be higher than the snippets that do not follow the rules.

6.6.2 Results: Eye Gaze Distribution and Preference Results

Rule R1 - Minimize Nesting We group the participants into two groups - one group that preferred to follow the minimize nesting rule and those that did not and show eye tracking metrics for each group separately. Figure 15 shows the average fixation counts and fixation durations by a participant's readability ranking. The participant ranked either the rule-following snippet on the left side or the rule-breaking snippet of the right side to be more readable when presented with a side-by-side method comparison task for readability of rule R1. The figure shows that participants that ranked the rule-breaking snippet to be more readable than the rule-following snippet for rule R1 had more fixations for longer periods throughout the entire task (averaging a 55% and 64% increase, respectively), especially on the rule-following snippet.

Table 17 presents results to determine whether there is a significant effect on the dependent variables based on whether or not readability rule R1 was followed. The participant responded with a ranking of whether the rule-following or rule-breaking snippet was more readable. The paired Wilcoxon signed-rank test was used to determine statistical significance for RQ5 as it was a paired comparison. Participants that rated the snippet that follows rule R1 to

Participant Ranking	**Rule-Following Snippet**	**Rule-Breaking Snippet**	Participant Ranking	**Rule-Following Snippet**	**Rule-Breaking Snippet**
Rule 1 Side-by-Side Fixation Counts			Rule 1 Side-by-Side Fixation Durations (seconds)		
Requirements	26.14	36.57	Requirements	6.36	8.70
Rule-Following Snippet	39.57	60.00	Rule-Following Snippet	15.08	29.87
Rule-Breaking Snippet	44.31	64.86	Rule-Breaking Snippet	12.89	20.40

Fig. 15 Average fixation counts and fixation durations by area of code and participants' ranking of a snippet (rule-following left versus rule-breaking right) for side-by-side method comparison tasks evaluating rule R1. Both snippets were logically correct

Table 17 Paired Wilcoxon signed-rank test results of all dependent variables for side-by-side method comparison tasks testing rule R1: minimize nesting

Preference	Metric	*p*	Cohen's *d*
Rule-Following	Vertical Next Text	*0.0014	0.5135
	Vertical Later Text	*0.0210	0.4205
	Horizontal Later Text	*0.0153	0.2197
	Regression Rate	*<0.0001	0.6972
	Line Regression Rate	*0.0002	0.6617
	Line Coverage	*0.0014	-0.7221
	Saccade Length	0.5497	-0.0774
	Fixation Counts	0.4843	-0.1305
	Fixation Durations	0.6137	-0.1185
Rule-Breaking	Vertical Next Text	0.3750	0.6816
	Vertical Later Text	0.2188	0.8820
	Horizontal Later Text	0.2188	0.8931
	Regression Rate	0.4688	0.5803
	Line Regression Rate	*0.0360	0.8938
	Line Coverage	0.3750	-0.2620
	Saccade Length	0.8125	-0.0909
	Fixation Counts	0.7344	-0.1444
	Fixation Durations	0.8125	-0.1151

Fixations are grouped by participants ranking either the rule-following snippet to the left or the rule-breaking snippet to the right higher
Significant comparisons are marked with an asterisk * before the p-value

be more readable than the rule-breaking snippet had a lower rate of gaze linearity metrics except for the line coverage with significant effect. A medium effect was observed in the rate of vertical next text ($p = 0.001$, Cohen's $d = 0.514$), regression rate ($p < 0.001$, Cohen's $d = 0.697$), line regression rate ($p < 0.001$, Cohen's $d = 0.662$) and line coverage ($p = 0.001$, Cohen's $d = -0.722$). A small effect was observed on vertical later text ($p = 0.021$, Cohen's $d = 0.421$) and horizontal later text ($p = 0.015$, Cohen's $d = 0.220$). However, participants that rated the rule-breaking snippet to be more readable showed fewer differences in eye movements between the two snippets, with the only significant effect being a lower line regression rate on the rule-breaking snippet with large effect ($p = 0.036$, Cohen's $d = 0.894$).

Finding: Participants that rated snippets that follow rule R1 as more readable had a lower rate of gaze linearity metrics on the rule-breaking snippet except line coverage which was higher.

Rule R2 - Avoid do-while Figure 16 shows the average fixation counts and fixation durations by a participant's readability ranking. The participant ranked either the rule-following snippet on the left side or the rule-breaking snippet of the right side to be more readable when presented with a side-by-side method comparison task for readability of rule R2. However, participants that rated the rule-breaking snippet to be more readable than the rule-following snippet for rule R2 had more fixations for longer periods throught the entire task (averaging a 46% and 39% increase, respectively).

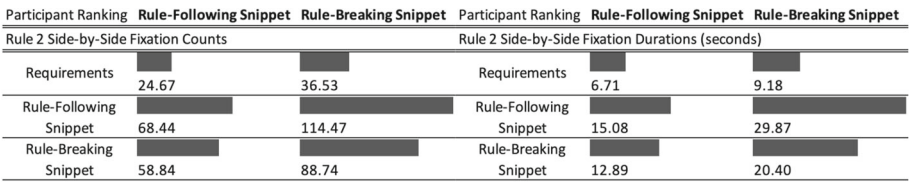

Fig. 16 Average fixation counts and fixation durations by area of code and participants' ranking of a snippet (rule-following left versus rule-breaking right) for side-by-side method comparison tasks evaluating rule R2. Both snippets were logically correct

Table 18 presents results to determine whether there is a significant effect on the dependent variables based on whether or not the readability rule R2 was followed and whether a participant rated the rule-following or rule-breaking snippet to be more readable. Table 18 shows that the right side rule-breaking snippet in a side-by-side method comparison task had a lower rate of horizontal later text, regression rate, and line regression rate among participants that rated the rule-following snippet to be more readable. However, line coverage was higher for the aforementioned participants. The horizontal later text had a small effect ($p = 0.026$, Cohen's $d = 0.375$). In addition, medium effects were observed with regression rate ($p = 0.001$, Cohen's $d = 0.706$), line regression rate ($p = 0.006$, Cohen's $d = 0.605$), and line coverage ($p = 0.007$, Cohen's $d = -0.674$).

Participants who rated the snippet that was rule-breaking to be more readable had a lower rate of vertical next text, regression rate, line regression rate, and a higher rate of line coverage

Table 18 Paired Wilcoxon signed-rank test results of fixation counts and durations of side-by-side method comparison tasks testing rule R2: avoid do-while

Preference	Metric	*p*	Cohen's *d*
Rule-Following	Vertical Next Text	0.0903	0.3807
	Vertical Later Text	0.2410	0.2791
	Horizontal Later Text	*0.0255	0.3751
	Regression Rate	*0.0010	0.7064
	Line Regression Rate	*0.0063	0.6051
	Line Coverage	*0.0069	-0.6740
	Saccade Length	0.7112	0.0540
	Fixation Counts	0.0577	0.2464
	Fixation Durations	0.0626	0.1964
Rule-Breaking	Vertical Next Text	*0.0033	0.9798
	Vertical Later Text	0.0061	0.8807
	Horizontal Later Text	0.0874	0.3377
	Regression Rate	*<0.0001	0.7341
	Line Regression Rate	*0.0017	0.8581
	Line Coverage	*0.0417	-0.4761
	Saccade Length	0.5678	-0.2933
	Fixation Counts	0.4565	0.3547
	Fixation Durations	0.3735	0.4293

Fixations are grouped by participants ranking either the rule-following snippet to the left or the rule-breaking snippet to the right higher
Significant comparisons are marked with an asterisk * before the p-value

when reading the snippet the rule-breaking snippet. A small effect was observed with the line coverage ($p = 0.042$, Cohen's $d = -0.476$), and a medium effect was observed with the regression rate ($p < 0.001$, Cohen's $d = 0.734$). Large effects were observed with vertical next text ($p = 0.003$, Cohen's $d = 0.980$) and line regression rate ($p = 0.002$, Cohen's $d = 0.858$).

Finding: Participants who rated snippets that follow R2 as more readable had a lower rate of horizontal later text on the rule-breaking snippet, whereas those that rated snippets that break rule R2 as more readable had a lower rate of vertical next text on the rule-breaking snippet. All participants had a lower rate of regression rate, line regression rate, and higher line coverage on the rule-breaking snippet, regardless of preference.

6.7 RQ6 Results: Secondary Factors: Native Language, Java Knowledge, and English Knowledge

We wanted to perform an exploratory analysis to see how this eye tracking study's results compare to the original online study. The online study showed native English speakers having shorter read and answer times than native Spanish speakers (12.29% faster and 31.06% faster, respectively). In lieu of Spanish speakers for this eye tracking study that replicates the online study, we have instead compared native English speakers to non-native English speakers, as there were no Spanish-speaking participants. The eye tracking study showed reversed results, where English speakers had longer read and answer times than non-native English speakers (8.1% slower and 1.2% slower, respectively), however we note that the difference is not as pronounced as in the online study.

When breaking participants down by Java knowledge, they were divided between those having less than 'Satisfactory' knowledge ('Poor' or 'Very Poor', 8.7% of participants) and those with at or more Java knowledge ('Satisfactory,' 'Good,' or 'Very Good', 91.3% of participants). The online study suggested that when Java knowledge increased, readability rating, multiple choice and logical correctness question accuracy, and comprehension confidence increased while read and answer time decreased. Results for the eye tracking study are shown in Fig. 17, which shows that as Java knowledge increased, the readability rating, comprehension confidence, logical correctness question accuracy, read time, and answer time decreased whereas multiple choice question accuracy increased. We refer the reader to Johnson et al. (2019) for a figure that is analogous to Fig. 17 in this paper. The only agreement between the two studies appears to be on the multiple choice question accuracy, and read time. There was partial agreement with answer time, where the online study shows a lower answer time with non-native English speakers (Spanish speakers) but a higher answer time with English speakers. This is in contrast to the eye tracking study, where both native and non-native English speakers both had lower answer times when their self-reported knowledge of Java was high.

We performed t-tests and found effect sizes to compare our results to the original study. When comparing the non-native English speakers against the overall dataset, most significant effects were similar except for the read times when comparing the snippets that do not follow rule R1 (minimize nesting), where there was no significant effect. Unlike the non-native English speakers, there were significant effects on lowering a native English speaker's ability to correctly determine whether a given snippet was logically correct with a medium effect when the snippet is logically incorrect for both rule R1 ($p = 0.002$, Cohen's $d = 0.679$) and rule R2 ($p = 0.001$, Cohen's $d = 0.704$), which was obfuscated in the complete data set.

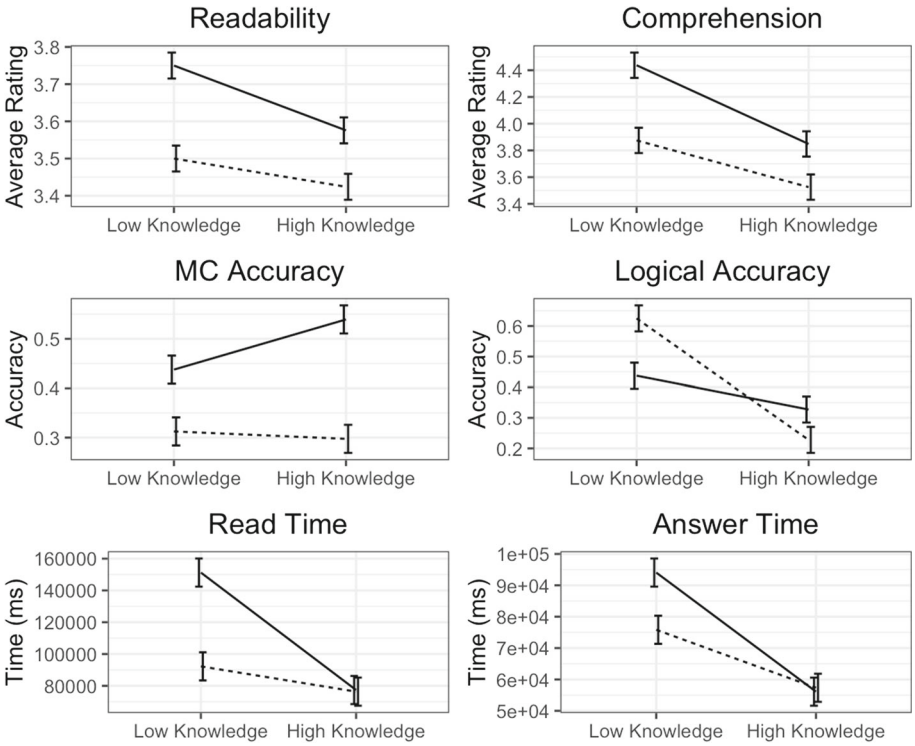

Fig. 17 Comparison of average metric values after breaking the population up into groups based on native language and self-reported knowledge of Java, for a total of 4 groups. Dotted lines represent non-native English speakers and solid lines represent native English speakers

When comparing the non-native English speakers with the native English speakers, non-native English speakers showed a significant effect on comprehension confidence on a rule-breaking snippet evaluating rule R1 (minimize nesting). On the other hand, non-native English speakers did not show any significant effects on the logical correctness question when a given snippet is logically correct regardless of whether it is evaluating rules R1 (minimize nesting) or R2 (avoid do-while).

Finding: Across all tasks, native English speakers were significantly less accurate in determining logical correctness in logically incorrect snippets for either readability rule compared to non-native English speakers.

6.8 Answer Correctness and Confidence

In the single method analysis tasks, participants were asked to answer a multiple choice question on the functionality of the code and a 5 point Likert scale question on how confident they are on the answer. We wanted to observe the differences in comprehension confidence between participants that answered the multiple choice comprehension question correctly and incorrectly. The average confidence rating of correct answers and incorrect across all rules was 3.684 and 3.310, respectively. When separated by rule, the average confidence rating for correct answers and incorrect answers for the minimize nesting rule was 3.953 and

3.568, respectively. Rule R2's confidence rating for correct and incorrect answers was 4.236 and 3.811, respectively. To see if these differences had any statistical signficance, we used the t-test to compare ratings. Whether the confidence rating was across all questions, rule R1, or rule R2, incorrectly answered questions had significantly lower confidence with a small effect ($p < 0.001$, Cohen's $d = 0.385$, $p = 0.001$, Cohen's $d = 0.489$, $p = 0.020$, Cohen's $d = 0.356$ respectively).

6.9 Post-Questionnaire Results

After participants finished all study tasks, they were asked to rank the importance of the readability rules R1 and R2 on a scale ranging from not important to very important (1-5). Participants generally rated rule R1: minimize nesting to be more important than rule R2: avoid do-while.

Out of 46 participants, 23 (50%) rated rule 1: minimize nesting to be very important, 21 (45.7%) participants rated the rule to be somewhat important, and 2 (4.3%) participants rated the rule to be neutral. No participants rated rule R1 to be not important. These results are comparable to the online study's results, where 133 out of 258 (51.6%) participants rated R1 to be very important, (36%) participants rated R1 to be somewhat important, 31 (12%) participants rated R1 to be neutral, and one participant (0.4%) rated R1 to be somewhat not important.

11 out of 46 (23.9%) participants rated rule R2: avoid do-while loops to be very important, 19 (41.3%) participants rated R2 to be somewhat important, 10 (21.7%) participants rated R2 to be neutral, 4 (8.7%) participants rated R2 to be somewhat not important, and 2 (4.3%) participants rated R2 to be not important. The results for rule R2 are also comparable to the online study's results where 44 out of 258 (17.1%) participants rated R2 to be very important, 85 (32.9%) rated R2 to be somewhat important, 88 (34.1%) participants rated R2 to be neutral, 20 (7.8%) participants rated R2 to be somewhat not important, and 21 (8.1%) participants rated R2 to be not important.

7 Threats to Validity

This section addresses the four main categories of threats to validity and ways we tried to mitigate them.

7.1 Internal Validity

In the online questionnaire-based study, there were potential threats to internal validity due to the fact that not all participants were supervised by a moderator throughout the study. To mitigate this, we instructed participants to refrain from using other tools or copy and paste the code to answer the questions. They were also asked to complete the study in one sitting. In the eye tracking study, no participants were able to use additional tools to understand the code used for the study or pause the experimental session as it was fully moderated and done in a controlled lab setting. Since the eye tracking study is a replication of the online study, the overall feasibility and time required to complete each of the tasks of the experiment have already been tested via initial pilots to avoid poorly designed experimental artifacts and to avoid fatigue effects on accuracy. Similar to the online questionnaire-based study, the participants were presented with the four problems in a different treatment in random order to

avoid any learning effects. There is a slight chance that the fixation correction method could have missed some fixations on lines. However, this risk is very low because we used chunks of fixations to move based on the shape of the line length. If fixations land on whitespace, we do not force them onto words on lines. The entire fixation trajectory is moved as one unit and never individual fixations.

7.2 External Validity

The code snippets selected for both studies are very short and isolated, tailored for the specific goals of the experiments. The results might be somewhat different if the code is extracted from the software artifacts of a real code base. Compared to the online study, there are fewer languages spoken by participants and the largest population of non-English speakers are natives of different languages (Spanish in the online study vs. Telugu in the eye tracking study). However, our replication shows similar measures to the online study discussed in Section 6.7. Further replications involving professional developers are still required as only a minority of our participants (11%) had experience working in industry, making our results difficult to generalize for professional software developers. The task type (multiple choice) and language chosen (Java) could also be potential threats to generalizing these results to other tasks (finding a bug, fixing a syntax problem, or adding a feature) and languages. Other replications with these variations in task and language are required.

7.3 Construct Validity

The experiment was designed to tease apart readability in correct and incorrect solutions to a problem. We did not have a formal specification of the problem. However, the natural language descriptions were reviewed to make sure they were not ambiguous via two small pilots done prior to doing the online-questionnaire study. It is quite possible that the readability of code can be affected by syntactic and semantic aspects and its presentation (indentation, font, colors, etc...). This is mitigated by choosing simple problems with short methods that do not use dependencies and comments and did not use colors. The only differences between the four solutions for a given problem are whether the solution follows a readability rule or is logically correct. Multiple choice questions related to method execution were used to determine level of understanding. They were asked about logical correctness as a separate Yes/No question. The read time was determined based on the time spent reading the code and is provided by the eye tracking equipment via time stamps for each fixation activity. For the preference tasks, we showed the rule-following snippet on the left and the rule-breaking snippet on the right for both rules. This could cause some threats to measurement however we can see from the results of RQ5, that participants read both the code snippets (line coverage and fixation counts were recorded for both rules on both snippets) and did not necessarily focus on the one on the left because the results also differ between rule R1 and R2 even though the rule-following snippet was always on the left. We did not tell the participants which snippet followed a rule or broke a rule. Finally, there is the potential threat of the participant changing behavior due to being observed (Hawthorne effect) in every study. For the eye tracking study, after the calibration was done, the work environment mimicked a code viewer. The study is as unobtrusive as we could make it as nothing comes into contact with the participant. The moderator although in the same room (quiet with no distractions) was not visible to the participant. We took every precaution to mimic the work environment of a developer to minimize this threat.

7.4 Conclusion Validity

We use standard statistical measures, namely the Student's t-test for timings and participant responses, and the paired Wilcoxon signed-rank test (due to low sample size) for eye tracking data and Cohen's d which are conventionally used in inferential statistics.

8 Discussion and Impact

The main findings of the eye tracking study conducted in this paper corroborate the results of the online study (Johnson et al. 2019), where following the minimize nesting rule has a significant impact on code comprehension while the avoid do-while rule did not have a significant impact on the metrics used for code comprehension. An overview of the findings for the online study and this eye tracking study (as measured in RQs 1 and 2) is shown in Table 19. Each dependent variable and the outcome from each study is shown in the table with the last column indicating if there was agreement between the two studies. For majority of the comprehension, confidence, and rating metrics of RQs 1 and 2, there was an agreement between the two studies as shown by the check mark in the last column of Table 19. For the ones that did not agree, the effect differences were very small (see third and fourth column of the table) between the two studies. Having a lower sample compared to the online-study might be the reason behind this difference. Refer to Table 20 that summarizes the eye tracking results for RQ3 and RQ4. Table 21 presents a summary of a pairwise comparison where the logically correct, rule following snippet is compared with three other categories (rule breaking + logically correct, rule following + logically incorrect, and rule breaking + logically incorrect). Both method analysis and method comparison tasks are shown.

From the secondary factors (see Section 6.7), the eye tracking study showed different results from the online study. In this study, we found that English speakers were significantly less accurate in determining logical correctness in logically incorrect snippets compared to non-native English speakers for either readability rule. However in the online study (Johnson et al. 2019), both English speakers and Spanish speakers were significantly less accurate in determining logical correctness in logically incorrect snippets (for either readability rule). In the eye tracking study, English speakers (who made up roughly half of the pool) had longer read and answer times than non-native English speakers (8.1% slower and 1.2% slower, respectively), however this was not significant. In the prior online study's analysis (Johnson et al. 2019), we split native Spanish speakers from native English speakers because the study pool involved many (73%) natively Spanish-speaking participants. For this study, we compare native English speaking with non-native English-speaking participants (not particularly Spanish). Most non-native participants listed Telugu as their native language. The discrepancy we see in the secondary factors read and answer times (albeit not significant) could be due to the difference in native languages between the two studies. With respect to significant findings in this study, native English speakers (∼50% of the pool) were significantly less accurate in determining logical correctness in logically incorrect snippets compared to non-native English speakers for both readability rules. In the online study, on the other hand, the English speakers (who made up 27% of the total pool) had significantly shorter read and answer times than any other group but most other metrics had similar effects. In both studies, we observe that with high knowledge of Java, the read time and answer time for both native and non-native speakers in both studies was reduced to almost no difference, whereas there was a difference in averages when comparing non-native and natives with low Java

Table 19 Comparison of findings between the online study and the eye tracking study

	Treatment	Online Study (Johnson et al. 2019)	Eye Tracking Study	Agreement
RQ1-Rule 1				
Level of Understanding	Following rule	↑ Very small effect on comprehension question	No significant effects	×
	Logically correct	↑ Small effect on comprehension question ↑ Medium effect on logical correctness question	↑ Large effect on comprehension question ↑ Small effect on logical correctness question	✓
Comprehension Time	Following rule	↑ Small effect on reading time	↑ Medium effect on reading time	✓
	Logically correct	↑ Very small effect on answer time	No significant effects	×
Readability Assessment	Following rule	↑ Large effect on perceived readability	↑ Large effect on perceived readability	✓
	Logically correct	No significant effects	No significant effects	✓
Comprehension Confidence	Following rule	↑ Small effect on confidence of understanding code	↑ Medium effect on confidence of understanding code	✓
	Logically correct	No significant effects	No significant effects	✓
Readability Rating		Minimize nesting rule is rated to be important	Minimize nesting rule is rated to be important	✓
RQ2-Rule 2				
Level of Understanding	Following rule	No significant effects	↑ Small effect on comprehension question	×
	Logically correct	↑ Medium effect on logical correctness question	↑ Small effect on logical correctness question	✓
Comprehension Time	Following rule	No significant effects	No significant effects	✓
	Logically correct	↑ Very small effect on answer time	No significant effects	×
Readability Assessment	Following rule	No significant effects	No significant effects	✓
	Logically correct	No significant effects	No significant effects	✓
Comprehension Confidence	Following rule	No significant effects	No significant effects	✓
	Logically correct	↓ Very small effect on confidence of understanding code	No significant effects	×
Readability Rating		Avoid do-while rule is rated to be important	Avoid do-while rule is rated to be important	✓

↑ denotes a positive effect when going from rule breaking to rule following while ↓ denotes a negative effect when going from rule breaking to rule following
RQ1 related to the minimize nesting rule and RQ2 related to the do-while rule

Table 20 Eye tracking metric results of the eye tracking study when doing a general comparison

	Task Type	Treatment	Eye Tracking Study
RQ3			
	Method Analysis	Rule Following	↓ Small effect on horizontal later text
			↓ Small effect on line regression rate
			↓ Very small effect on fixation counts
		Logically Correct	↑ Small effect on horizontal later text
			↑ Small effect on regression rate
			↑ Small effect on line regression rate
			↑ Very small effect on fixation counts
	Method Comparison	Rule Following	↓ Medium effect on line regression rate
			↑ Very large effect on line coverage
RQ4			
	Method Analysis	Rule Following	↑ Very small effect on fixation counts
			↓ Very small effect on fixation durations
		Logically Correct	↑ Very small effect on fixation counts
	Method Comparison	Rule Following	↓ Medium effect on vertical next text
			↓ Medium effect on vertical later text
			↓ Small effect on horizontal later text
			↓ Medium effect on regression rate
			↓ Medium effect on line regression rate
			↑ Medium effect on line coverage

↑ denotes a positive effect when going from rule breaking to rule following while ↓ denotes a negative effect when going from rule breaking to rule following
RQ3 related to the minimize nesting rule and RQ4 related to the do-while rule

knowledge with the native English participants taking longer to read and answer but also answering much more accurately than non-native speakers. In other comparisons, we notice that the averages for readability, comprehension, and logical accuracy are much closer to

Table 21 Eye tracking metric results of the eye tracking study when doing a pairwise comparison

	Task Type	Treatment	Eye Tracking Study
RQ3			
	Method Analysis	Rule Following	↓ Very small effect on fixation counts
			↓ Small effect on fixation durations
		Logically Correct	No significant effects
		Both	No significant effects
RQ4			
	Method Analysis	Rule Following	↑ Small effect on fixation counts
		Logically Correct	↑ Small effect on fixation counts
		Both	↑ Small effect on fixation counts

↑ denotes a positive effect when going from rule breaking to rule following while ↓ denotes a negative effect when going from rule breaking to rule following. RQ3 related to the minimize nesting rule and RQ4 related to the do-while rule

each other for native and non-native speakers with high knowledge of Java. The only case where this does not hold is for the multiple choice accuracy question where the gap between native and non-native speakers is higher with native speakers performing way better with higher knowledge of Java.

We now briefly highlight the results for the minimize nesting rule and the avoid do-while rule before discussing the implications for educators and practitioners.

8.1 Observations on the Minimize Nesting Rule

For the level of understanding of rule R1: minimize nesting, there was a small decrease (5.4%) in question accuracy rates when a given snippet was following the rule. However, there was no statistical significance. On the other hand, the eye tracking study did find logically correct snippets to have a larger effect on question accuracy rates (3.8 times) for comprehension questions than the question asking about a snippet's logical correctness. While similar effects were found in the online study, logically correct questions had a higher effect on the logical correctness question than the accuracy of finding the bug. The eye tracking study also found that whether or not the minimize nesting rule was followed had no effect on the accuracy of answering either question, whereas the online study displayed a slight increase in accuracy for the comprehension question when the rule was followed.

For the comprehension time of rule R1 (minimize nesting), the eye tracking study corroborates with the online study as shown in Fig. 3 and Table 19, where logically correct snippets consistently take roughly 7.1% less time to read while the logically incorrect, rule-following snippet (R_1L_0) takes the longest time to answer.

Readability assessments for rule R1 also show that following the rule made the readers perceive the code as more readable regardless of its logical accuracy. Readability ratings for side-by-side method comparison tasks also indicated that snippets following the minimize nesting rule were ranked higher in readability by the vast majority of participants (84.09%). Comprehension confidence, or the measure of how well a reader believes they understood a given snippet, for rule R1 was 14.8% higher in snippets following the minimize nesting rule.

Our findings for eye tracking metrics on snippets for rule R1 (see Table 20), the minimize nesting rule, show that for the single method analysis tasks, snippets that did not follow the readability rule had on average 3.5% lower number of fixations than snippets that followed the readability rule. However, logically incorrect snippets had on average 3.9% more fixations. A higher number of fixations on incorrect snippets could mean that the participants were trying to understand it more with respect to the prompt. These differences contrast what was indicated by the participants as they perceived rule-breaking snippets to be more difficult to read. When we performed our pairwise comparisons (see Table 21), snippets that broke rule R1 had a lower rate of fixation counts and fixation durations, suggesting there was less movement throughout the code. This could imply that the rule following snippets induced the participants to spend more time on understanding the well formatted code.

For the side-by-side method comparison tasks (see Table 20 - Method Comparison row), a smaller line regression rate but higher line coverage was observed when participants looked at the rule-breaking snippet. This indicates that more lines are looked at when looking at a rule-breaking snippet indicating perhaps an inefficient search.

8.2 Observations on the Avoid Do-While Rule

Our findings show that neither rule-following nor logical correctness had any significant effect on the level of understanding, comprehension time, readability assessment, and comprehension confidence for the avoid do-while rule R2. However, in side-by-side method comparison tasks, a majority of the participants (56.82%) did rate the snippet following the avoid do-while rule as being more readable than the snippet that did not follow the rule. This may mean that following rule R2 is less important for perceived readability.

Eye tracking metrics on rule R2 (see Table 20), the avoid do-while rule, show that for the single method analysis tasks, participants had a higher number of fixations and lower fixation durations on the rule-breaking snippets. Logically incorrect snippets also had a higher number of fixations. The higher number of fixations in the snippets that did not avoid using do-while loops could indicate that participants looked at the lines more perhaps to grasp the logic of do-while loop but the time for the visits was short. No significant effects were observed on other gaze-based metrics.

When we performed pairwise comparisons for snippets evaluating rule R2 (see Table 21), higher fixation counts were observed when a snippet was either rule-breaking, logically incorrect, or both. This indicates that a lot more gaze visits are required if snippets used do-while and if they were incorrect.

For the side-by-side method comparison tasks (see Table 20 - Method Comparison row), participants did not show a difference in fixation counts and durations when looking at the rule-breaking snippet. However, all gaze-based linearity metrics except the line coverage (which was higher) and saccade length (which had no difference) were significantly lower when the snippet was rule-breaking.

8.3 Implications and Research Directions

One of the reasons why we conducted the replication of the Johnson et al. (2019) study using an eye tracker, was to observe if there was any difference in how much time developers spent reading and navigating through the code snippets on the line level in two different readability variants for two different rules as suggested by Boswell and Foucher (2011). In order to do the comparisons, we use several eye tracking specific metrics as listed in Table 8 - most important of which were the fixation counts, fixation durations, and saccade lengths. The research literature on cognitive load (Debue and van de Leemput 2014; Godwin et al. 2021) suggests that longer fixations and shorter saccades are both associated with higher cognitive load. Our results indicate that when snippets followed the minimize nesting rule, they resulted in lower fixation counts and shorter fixation durations. However when we normalized for snippet length, the rule-breaking snippet had lower fixations. A logically incorrect snippet had higher regressions (re-reading of lines) indicating difficulty in understanding or possibly trying to figure out the problem. We also report higher fixation counts for logically incorrect snippets that did not follow the minimize nesting rule. This result also translated to participants spending less time on tasks when the minimize nesting rule was followed (no significant differences in accuracy were reported in the eye tracking study). Note that the participants did not know if the code was incorrect nor did they know whether the snippet followed a rule or not. All of the above results indicate that following the minimize nesting rule results in lower cognitive load for developers.

Results from the avoid do-while rule were not as pronounced. Snippets that do not follow the avoid do-while rule have higher fixation counts and shorter fixation durations but otherwise do not have any effect on how the code was navigated. This was a surprising result as the results are opposite to what we found for the minimize nesting rule. We did however find that when participants followed the avoid do-while loop they were significantly more accurate in the tasks (no differences in time were found). As mentioned earlier, these differences did not translate to any differences in speed (see Table 10). The eye tracking results seem to indicate that following the avoid do-while rule results in higher cognitive load because this group of participants had higher fixation counts, longer fixation durations, and shorter saccades. They also, however, had significantly higher accuracy compared to when the avoid do-while rule was not followed. None of the other eye tracking metrics had any major effect between the rule following or rule breaking snippets for the avoid do-while rule either. This could indicate that developers really do not care about avoiding the do-while rule and it does not bother them as much as following the minimize nesting rule. In the side-by-side ratings, overall our participants ranked the snippets that followed both readability rules higher than the ones that did not follow the rules. So even though they ranked the rules as important overall, their eye movements tended to support the ranking of the minimize nesting rule more than the avoid do-while rule.

We discuss the implications of these results to educators and practitioners. Code readability is rarely talked about as a front-and-center topic while teaching programming. A lot of emphasis is put on functionality (with good reason), however this could have some unwanted productivity issues in the long run. If code is not written with the goal of being readable not just by the person who wrote it but also by anyone else who needs to understand the code, the person might spend much more time than necessary. We show this to be true when the minimize nesting rule is not followed. We also show that the cognitive load to understand the program increased for developers reading the snippet that did not follow the minimize nesting rule. Most educators don't consider readability a main topic but rather gloss over it as a secondary issue that is looked at only after the code works. However, we have seen from these two studies, Johnson et al. (2019) and this study, that having readable code makes developers more efficient. If you are more efficient, you save time in the long run. Teaching code readability as part of the Computer Science and Software Engineering curricula is crucial for students to learn how to write readable code. Making code readability part of the rubric to be graded and teaching it as part of learning to program is crucial. One of the main factors that needs to be addressed is to come up with a clear definition of what readability means.

It is worth mentioning that in contrast, in industrial settings the concern for readability is becoming increasingly important, as evidenced by the coding standards defined by companies or developer communities. For example, "The Google Java Style Guide" (Google Java Style Guide 2023) and "The PEP 8 - Style Guide for Python Code" (PEP 8 2023) are full of explicit coding recommendations aimed at making code easy to understand and easy to change. A plausible explanation for this contrast is that students rarely face the problems inherent in maintaining and evolving real codebases, while professional developers do.

Practitioners typically adhere to some coding style as dictated by their workplace. Some of these coding styles have readability built into them in the form of pretty printing or linters that must be run before the code is checked into the repository. Code reviews are also typically performed where an assigned reviewer checks to see if the code meets certain standards for

logic and flow. Having checks at this stage for logic simplification such as minimizing nesting or avoid do-while would be worthwhile. Results of the study presented in this article showed that following the minimize nesting rule took lower time/effort whereas following the avoid do-while rule had higher performance accuracy. These time savings directly relate to cost savings for developers eventually making them more productive.

We believe readability is not really well defined in the literature and could typically have different definitions based on whom you ask. The ground truth for what makes code readable does not exist at this time. One way forward would be to conduct a survey and/or interview with developers/stakeholders in different roles to determine what readability means to them. Does indentation and color play a role? or should the semantics be more of the focus, like we did in this study. We cannot automate solutions to readability if we do not first understand how to define readability and when and what context it is most useful to various categories of developers. *We strongly believe that code should be written keeping the developer who reads it in mind. In other words it needs to be developer-centric.* Currently, code for the most part, is written in a purely artifact-centric way i.e., without keeping the developer who will read it in mind. As developers, do we think about how someone might feel after they read our code? In order to make writing code more developer-centric, as a community, we need to ask the consumers of code i.e., mainly active developers in various languages, how they perceive readability. These ventures can take the form of traditional questionnaire-based surveys, but also interviews, eye tracking studies, as well as ethnographic studies might be valuable in learning more details as to the why certain code is more readable than others.

Boswell et al. define a fundamental theorem of readability (Boswell and Foucher 2011). They state that code should be written to minimize the time it takes someone else to understand it. They further describe that to fully understand the code, one should be able to change it, fix bugs, and describe how it fits into the rest of the system. Boswell and Foucher discuss surface-level improvements vs. simplifying logic. Surface level improvements are related to naming, comments, indentation, use of color schemes, pretty printing, placement of blank lines to delineate chunks of code, to name a few. Simplifying logic refers to making control flow easier to follow (minimize nesting rule), order of the if/else statements, avoiding the do-while, returning early from a function, ordering arguments in functions, to name a few. This list by Boswell and Foucher is a good start however more nuances might exist. For example, naming might not necessarily be a surface level improvement if it also simplifies the logic. Coming up with good variable names is hard and not trivial (Alsuhaibani et al. 2021).

We also know from prior literature that experts read code differently from novices (Busjahn et al. 2015). Experts generally have shorter fixations on source code and read the code as if it were being executed compared to novices who read the code in a story-like fashion i.e., typically top-to-down and left-to-right. Bauer et al. showed that the level of indentation did not affect performance (Bauer et al. 2019). Oliveira on the other hand observed that appropriate indentation, end block delimiters, and restricting the code to 80 characters had a positive impact on code readability (Oliveira et al. 2023). More larger-scale realistic studies are needed to address these mixed results. With respect to pretty printing and layouts, Siegmund et al. found that disrupting the layout did not affect comprehension performance when tested using fMRI (Siegmund et al. 2017). These studies are done on very small code snippets (current study included). We have shown in prior work that results from small code snippets do not necessarily transfer to studies using real-world code snippets (Abid et al. 2019). For this reason, more studies need to be conducted in realistic settings to build comprehensive theories on readability. This paper seeks to call attention to the community to gather data on the various aspects of what readabilty means.

8.4 The Role of Eye Tracking in Future Code Readability Studies

This study advances the understanding of how developers read code written for two specific code readability rules. By using eye tracking metrics at the line level, we can objectively measure navigation patterns across lines and identify which lines developers spend more time reading. Taking a developer-centric approach to code reading, as measured by eye gaze, provides crucial evidence for the effectiveness of readability rules. The use of eye tracking to measure code readability boasts high construct validity, as it directly correlates to how developers read code. The metrics derived from eye gaze offer the most accurate insight into whether what developers see affects their comprehension of readable code. Linearity metrics, in particular, provide objective measures of gaze navigation across the code, which cannot be captured in online questionnaire-based studies. The need for an eye tracker, in conjunction with questionnaire-based studies, depends on the researcher's goals. If the aim is to understand how developers read code and investigate navigation patterns between lines, an eye tracker is essential. Neither questionnaire-based studies nor keystroke and interaction data can provide this level of insight, as demonstrated by our previous study (Kevic et al. 2015), which simultaneously collected data from an eye tracker and interactions, highlighting the significance of gaze context.

While questionnaire-based studies have their limitations, they are not entirely without value. One approach to bolster their findings is to validate them through replication using eye tracking, which can provide further confirmation or refutation of theories about reading patterns. By overlaying gaze patterns onto questionnaire-based data, we can gain a deeper understanding of the results and uncover additional evidence. However, it is essential to recognize that differences in tasks, demographics, or other factors may lead to varying results across studies. Therefore, replication with diverse demographics and tasks is crucial.

As the body of research grows, meta-analyses can be conducted to identify commonalities and differences between studies. Furthermore, it is important to avoid making sweeping conclusions based on a single study's findings. Due to various factors such as differences in tasks or demographics, we might not always see the same effect across studies. We also cannot conclude that if a measure is not significant for one particular task or language, it would not be significant for a different task or language as shown by Mansoor et al. (2024). In addition to significance, another thing to keep in mind is the effect size. Significance with a large effect is better than significance with a small effect because *the magnitude conveys the practical significance of the results*. Reporting effect sizes is vital for conducting meta-analyses across studies and for informing future research priorities.

We do not claim to have solved the code readability problem in this paper. There is much more work to do in this area including defining what readability actually means as it could mean different things to different stakeholders. We revisit the question we posed in the Introduction about what makes code readable and how to verify readability. This study, in part, contributes to understanding what makes code readable and how we could verify readability via eye gaze patterns. We believe this study investigates the structural aspect of code and its importance to understanding code readability.

9 Conclusions and Future Work

The paper presents one of the first eye tracking studies to investigate the impact of two readability rules namely, minimize nesting and avoid do-while loops, on source code program

comprehension. The 46 participants performed four method analysis tasks evaluating the minimize nesting rule (R1) and another four evaluating the avoid do-while rule (R2) for a total of eight method analysis tasks. The second part of the study then had each person complete two method side-by-side comparison tasks, one for each rule.

Results for the minimize nesting rule gave similar results to the original study (Johnson et al. 2019), where following the rule results in developers spending less time reading the source code in order to understand it and also increased the confidence of the developer's answer and their accuracy. However, from the eye tracking data in this study, we also found that participants had fewer fixations per second on snippets that violated the rule over the snippets that adhered to the rule. The higher fixation counts indicate a higher cognitive load when the minimize nesting rule was followed with higher accuracy levels.

Results for the avoid do-while rule suggest the rule is largely determined by personal preference of the individual with no significant effect on the time spent or how understandable the code was perceived. However, when the avoid do-while rule was followed, it resulted in higher accuracy in performance Compared to the online study (Johnson et al. 2019), fewer participants ranked the method following the rule as more readable, but it was still a majority. Participants had more fixations and less time on snippets that violated the rule over the snippets that adhered to the avoid do-while rule.

The perceived readability of both coding practices indicates following them is worth the effort. In the case of rule R1: minimize nesting, 84.09% of participants judged the snippet following the rule to be more readable compared to the same problem that does not follow the rule. In the case of rule R2: avoid do-while, 56.82% of participants judged the snippet following the rule to be more readable compared to the same problem that does not follow the rule.

As part of future work, there needs to be a clear definition of what readability means by soliciting feedback via interviews from different stakeholders. Furthermore, other rules in larger more realistic contexts will be evaluated using eye tracking with derived gaze metrics (Sharafi et al. 2015) at the token-level. We plan on conducting further eye tracking studies in larger codebases and realistic code settings using modern IDEs such as Visual Studio and the iTrace eye tracking infrastructure (Guarnera et al. 2018) to further provide evidence on code readability.

Acknowledgements We would like to thank all the participants in this study and Alex Hoffman for insightful discussions on statistical significance reporting. This work was funded in part by the US National Science Foundation under grant numbers CCF 18-55756 and CNS 18-55753.

Data Availability The experimental data and results supporting this study's findings are available in OSF with the identifier https://osf.io/m39p8 (Park et al. 2023).

Declarations

Conflicts of interest The authors declared that they have no conflict of interest.

Ethical standard The study was approved by the IRB review board at Youngstown State University.

References

Abid, NJ, Sharif, B, Dragan, N, Alrasheed, H, Maletic JI (2019) Developer reading behavior while summarizing Java methods : Size and context matters. In: Proceedings of the 41th International Conference on Software Engineering, ICSE 2019, page To Appear, New York, NY, USA, 2019. ACM

Ajami S, Woodbridge Y, Feitelson DG (2017) Syntax, predicates, idioms - what really affects code complexity? In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), pages 66–76

Alaboudi, A., LaToza TD (2021) Edit - run behavior in programming and debugging. In: 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), pages 1–10

Alsuhaibani RS, Newman CD, Decker MJ, Collard, ML, Maletic JI (2021) A survey on method naming standards: Questions and responses artifact. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), pages 242–243

Andersson R, Larsson L, Holmqvist K, Stridh M, Nyström M (2017) One algorithm to rule them all? An evaluation and discussion of ten eye movement event-detection algorithms. Behav Res Methods 49(2):616–637

Avidan, E, Feitelson DG (2017) Effects of variable names on comprehension: An empirical study. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), pages 55–65

Barbosa LF, Pinto VH, de Souza ALOT, Pinto G (2022) To what extent cognitive-driven development improves code readability? In: Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '22, page 238-248, New York, NY, USA. Association for Computing Machinery

Bauer J, Siegmund J, Peitek N, Hofmeister JC, Apel S (2019) Indentation: Simply a matter of style or support for program comprehension? In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pages 154–164

Beelders TR, du Plessis J-PL (2015) Syntax highlighting as an influencing factor when reading and comprehending source code. Journal of Eye Movement Research 9(1)

Benjamini Y, Hochberg Y (1995) Controlling the false discovery rate: A practical and powerful approach to multiple testing. Journal of the Royal Statistical Society. Series B (Methodological), 57(1):289–300

Binkley D, Davis M, Lawrie D, Maletic JI, Morrell C, Sharif B (2013) The impact of identifier style on effort and comprehension. Empir Softw Eng 18(2):219–276

Börstler J, Caspersen ME, Nordström M (2016) Beauty and the beast: On the readability of object-oriented example programs. Software Qual J 24(2):231–246

Börstler J, Störrle H, Toll D, van Assema J, Duran R, Hooshangi S, Jeuring J, Keuning H, Kleiner C, MacKellar B (2018) "I know it when I see it" perceptions of code quality: ITiCSE '17 working group report. In: Proceedings of the 2017 ITiCSE Conference on Working Group Reports, ITiCSE-WGR '17, page 70-85, New York, NY, USA. Association for Computing Machinery

Boswell D, Foucher T (2011) The Art of Readable Code. O'Reilly Media, Inc

Brooks R (1983) Towards a theory of the comprehension of computer programs. Int J Man Mach Stud 18(6):543–554

Buse RP, Weimer WR (2008) A metric for software readability. In: Proceedings of the 2008 International Symposium on Software Testing and Analysis, ISSTA '08, page 121-130, New York, NY, USA. Association for Computing Machinery

Buse RPL, Weimer WR (2010) Learning a metric for code readability. IEEE Trans Software Eng 36(4):546–558

Busjahn T, Bednarik R, Begel A, Crosby M, Paterson JH, Schulte C, Sharif B, Tamm S (2015) Eye movements in code reading: Relaxing the linear order. In: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, pages 255–265

Cates R, Yunik N, Feitelson DG (2021) Does code structure affect comprehension? on using and naming intermediate variables. In: 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), pages 118–126

Daka E, Campos J, Fraser G, Dorn J, Weimer W (2015) Modeling readability to improve unit tests. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, page 107-118, New York, NY, USA, 2015. Association for Computing Machinery

Debue N, van de Leemput C (2014) What does germane load mean? an empirical contribution to the cognitive load theory. Front Psychol 5

Dorn J (2012) A general software readability model

dos Santos RMA, Gerosa MA (2018) Impacts of coding practices on readability. In: Proceedings of the 26th Conference on Program Comprehension, pages 277–285

Fakhoury S, Roy D, Hassan A, Arnaoudova V (2019) Improving Source Code Readability: Theory and Practice. In: 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), pages 2–12

Flesch R (1948) A new readability yardstick. J Appl Psychol 32(3):221–233

Gill GK, Kemerer CF (1991) Cyclomatic complexity density and software maintenance productivity. IEEE Trans Software Eng 17(12):1284–1288

Godwin HJ, Hout MC, Alexdóttir KJ, Walenchok SC, Barnhart AS (2021) Avoiding potential pitfalls in visual search and eye-movement experiments: A tutorial review. Atten Percept Psychophys 83(7):2753–2783

Google Java Style Guide. https://google.github.io/styleguide/javaguide.html. Accessed: December 19, 2023

Guarnera DT, Bryant CA, Mishra A, Maletic JI, Sharif B (2018) itrace: Eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, ETRA '18, pages 105:1–105:3, New York, NY, USA. ACM

Hunter-Zinck H, de Siqueira AF, Vásquez VN, Barnes R, Martinez CC (2021) Ten simple rules on writing clean and reliable open-source scientific software. PLOS Computational Biology 17(11):1–9

Jbara A, Matan A, Feitelson DG (2012) High-mcc functions in the linux kernel. In: 2012 20th IEEE International Conference on Program Comprehension (ICPC), pages 83–92

Johnson J, Lubo S, Yedla N, Aponte J, Sharif B (2019) An empirical study assessing source code readability in comprehension. In: IEEE ICSME, pages 513–523

Kevic K, Walters BM, Shaffer TR, Sharif B, Shepherd, DC, Fritz T (2015) Tracing software developers' eyes and interactions for change tasks. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, pages 202–213, New York, NY, USA . ACM

Lakshmanan KB, Jayaprakash S, Sinha PK (1991) Properties of control-flow complexity measures. IEEE Trans Software Eng 17(12):1289–1295

Lawrie D, Morrell C, Feild H, Binkley D (2006) What's in a name? a study of identifiers. In: 14th IEEE International Conference on Program Comprehension (ICPC'06), pages 3–12

Ljung K, Gonzalez-Huerta J (2022) "to clean code or not to clean code" a survey among practitioners. In: Taibi D, Kuhrmann M, Mikkonen T, Klünder J, Abrahamsson P (eds) Product-Focused Software Process Improvement. Cham. Springer International Publishing, pp 298–315

Mannan UA, Ahmed I, Sarma A (2018) Towards understanding code readability and its impact on design quality. In: Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering, NL4SE 2018, page 18-21, New York, NY, USA . Association for Computing Machinery

Mansoor N, Peterson CS, Dodd MD, Sharif B (2024) Assessing the effect of programming language and task type on eye movements of computer science students. ACM Trans Comput Educ 24(1):2:1–2:38

Martin R (2009) Clean Code - A Handbook of Agile Software Craftmanship. Prentice Hall

Mi Q, Chen M, Cai Z, Jia X (2023) What makes a readable code? a causal analysis method. Software: Practice and Experience 53:1–19

Mi Q, Hao Y, Ou L, Ma W (2022) Towards using visual, semantic and structural features to improve code readability classification. J Syst Softw 193:111454

Mi Q, Keung J, Xiao Y, Mensah S, Mei X (2018) An inception architecture-based model for improving code readability classification. In: Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018, EASE'18, page 139-144, New York, NY, USA. Association for Computing Machinery

Miara RJ, Musselman JA, Navarro JA, Shneiderman B (1983) Program indentation and comprehensibility. Commun ACM 26(11):861–867

Minelli R, Mocci A, Lanza M (2015) I know what you did last summer - an investigation of how developers spend their time. In: 2015 IEEE 23rd International Conference on Program Comprehension, pages 25–35

Obaidellah U, Al Haek M, Cheng PC-H (2018) A survey on the usage of eye-tracking in computer programming. ACM Comput Surv 51(1)

Oliveira D, Bruno R, Madeiral F, Castor F (2020) Evaluating code readability and legibility: An examination of human-centric studies. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), pages 348–359

Oliveira D, Santos R, Madeiral F, Masuhara H, Castor F (2023) A systematic literature review on the impact of formatting elements on code legibility. J Syst Softw 203:111728

Olsen A (2012) The Tobii I-VT fixation filter. Tobii Technology 21:4–19

Park K, Weill-Tessier P, Brown N, Sharif B, Jensen N, Kölling M (2023) An eye tracking study assessing the impact of background styling in code editors on novice programmers' code understanding. In: 19th ACM Conference on International Computing Education Research (ICER)

Park KI, Sharif B, Johnson J (2023) An eye tracking study assessing code readability rules in program comprehension - replication package. https://www.osf.io/m39p8/. Accessed: December 19, 2023

PEP 8 - Style Guide for Python Code | peps.python.org. https://peps.python.org/pep-0008/. Accessed: December 19, 2023

Peterson CS, Park K-I, Baysinger I, Sharif B (2021) An eye tracking perspective on how developers rate source code readability rules. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), pages 138–139

Piantadosi V, Fierro F, Scalabrino S, Serebrenik A, Oliveto R (2020) How does code readability change during software evolution? Empirical Softw Engg 25(6):5374–5412

Posnett D, Hindle A, Devanbu P (2011) A simpler model of software readability. In: Proceedings of the 8th Working Conference on Mining Software Repositories, MSR '11, pages 73–82, New York, NY, USA. ACM

Posnett D, Hindle A, Devanbu P (2021) Reflections on: A simpler model of software readability. ACM SIGSOFT Software Engineering Notes 46(3):30–32

Scalabrino S, Bavota G, Vendome C, Linares-Vásquez M, Poshyvanyk D, Oliveto R (2017) Automatically assessing code understandability: How far are we? In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pages 417–427

Scalabrino S, Bavota G, Vendome C, Linares-Vásquez M, Poshyvanyk D, Oliveto R (2021) Automatically assessing code understandability. TSE 47(3):595–613

Scalabrino S, Linares-Vásquez M, Oliveto R, Poshyvanyk D (2018) A comprehensive model for code readability. Journal of Software Systems 30(6):e1958

Scalabrino S, Linares-Vásquez M, Poshyvanyk D, Oliveto R (2016) Improving code readability models with textual features. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*, pages 1–10

Schankin A, Berger A, Holt DV, Hofmeister JC, Riedel T, Beigl M (2018) Descriptive compound identifier names improve source code comprehension. In: Proceedings of the 26th Conference on Program Comprehension, pages 31–40

Sedano T (2016) Code Readability Testing, an Empirical Study. In: 2016 IEEE 29th International Conference on Software Engineering Education and Training (CSEET), pages 111–117

Sharafi Z, Shaffer T, Sharif B, Guéhéneuc Y (2015) Eye-tracking metrics in software engineering. In: Sun J, Reddy YR, Bahulkar A, Pasala A (eds) 2015 Asia-Pacific Software Engineering Conference, APSEC 2015, New Delhi, India, December 1–4, 2015. IEEE Computer Society, pp 96–103

Sharafi Z, Soh Z, Guéhéneuc Y-G (2015) A systematic literature review on the usage of eye-tracking in software engineering. Inf Softw Technol 67:79–107

Siegmund J, Peitek N, Parnin C, Apel S, Hofmeister J, Kästner C, Begel A, Bethmann A, Brechmann A (2017) Measuring neural efficiency of program comprehension. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, page 140-150, New York, NY, USA . Association for Computing Machinery

Smith EA, Kincaid JP (1970) Derivation and validation of the automated readability index for use with technical materials. Hum Factors 12(5):457–564

Storey M-A (2005) Theories, methods and tools in program comprehension: past, present and future. In: 13th International Workshop on Program Comprehension (IWPC'05), pages 181–191

Wiese ES, Rafferty AN, Fox A (2019) Linking code readability, structure, and comprehension among novices: It's complicated. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET), pages 84–94

Winkler D, Urbanke P, Ramler R (2022) What do we know about readability of test code? - a systematic mapping study. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pages 1167–1174

Xia X, Bao L, Lo D, Xing Z, Hassan AE, Li S (2018) Measuring program comprehension: A large-scale field study with professionals. IEEE Trans Software Eng 44(10):951–976

Kang-il Park is a Ph.D. student in Computer Science at the University of Nebraska-Lincoln (UNL) in Lincoln, Nebraska under the supervision of Dr. Bonita Sharif. Kang-il received his M.S. in Computer Science from UNL in 2020 and his B.S. in Computer Science at the University of South Dakota in 2017. His research interests are in eye tracking, software engineering, and program comprehension.

Jack Johnson is a Ph.D. student in Computer Science at the University of Minnesota Twin Cities (UMN) in Minneapolis, Minnnesota under the supervision of Dr. Mattia Fazzini. Jack received his M.S. in Computer Science from UMN in 2024 and his B.S. in Computer Science from UNL in 2019. He worked as a software engineer for IBM from 2019 to 2022. His research interests are in software testing, mobile development, and applied machine learning.

Cole S. Peterson graduated with her Masters in Computer Science from the University of Nebraska at Lincoln in 2020 under the supervision of Dr. Bonita Sharif. She currently works as a software engineer at a local company in Lincoln. Her research interests are in software engineering, eye tracking, polyglot programming studies, and program comprehension.

Nishitha Yedla holds a Master's degree in Computer Science from Youngstown State University. While studying, Nishitha served as a graduate assistant assisting Dr. Sharif on code readability research. She is now working as a Senior Full Stack Engineer at Fidelity Investments.

Isaac Baysinger is a DevSecOps Software Engineer at VIAVI Solutions in Minneapolis, MN, where he works with network performance monitoring and diagnostic products. Isaac received his B.S. in Computer Science from the University of Nebraska-Lincoln in 2023, where he assisted in eye-tracking and source code readability research under Dr. Bonita Sharif.

Jairo Aponte is an Associate Professor in the Department of Computing Systems and Industrial Engineering at Universidad Nacional de Colombia. He earned an Engineering and a Master degree in Computing Systems Engineering from the Universidad de los Andes (Bogotá), and a Ph.D. in Engineering from Universidad Nacional de Colombia (Bogotá). His current research interests are in software evolution and maintenance, processes and tools for software development, program comprehension and human aspects of software engineering.

Bonita Sharif is an Associate Professor in the School of Computing at University of Nebraska at Lincoln (UNL), Lincoln, Nebraska USA. She received her Ph.D. in 2010 and MS in 2003 in Computer Science from Kent State University, U.S.A and B.S. in Computer Science from Cyprus College, Nicosia Cyprus. Her research interests are in eye tracking related to software engineering, empirical software engineering, program comprehension, emotional awareness, software traceability, and software visualization to support maintenance of large systems. She serves on numerous program committees including ICSE, ASE, ESEC/FSE, ICSME, VISSOFT, SANER, and ICPC. She served as general chair of VISSOFT 2016 and ETRA 2018 and 2019. She served as program chair for ICPC 2023 technical track. She is an associate editor for Transactions on Computing Education and for Transactions of Software Engineering. She is also the Steering Committee Chair for the ACM Symposium on Eye Tracking Research and Applications. Sharif is a recipient of the NSF CAREER award and the NSF CRI award related to empowering software engineering with eye tracking. She also received the NCWIT Undergraduate Student Mentoring award in 2016. She directs the Software Engineering Research and Empirical Studies Lab at UNL's School of Computing.

Authors and Affiliations

Kang-il Park[1] · Jack Johnson[2] · Cole S. Peterson[1] · Nishitha Yedla[3] · Isaac Baysinger[1] · Jairo Aponte[4] · Bonita Sharif[1]

✉ Bonita Sharif
bsharif@unl.edu

Kang-il Park
kangil.park@huskers.unl.edu

Jack Johnson
joh19267@umn.edu

Cole S. Peterson
Cole.Scott.Peterson@huskers.unl.edu

Nishitha Yedla
nyedla@student.ysu.edu

Isaac Baysinger
isaacbaysinger@huskers.unl.edu

Jairo Aponte
jhapontem@unal.edu.co

1 University of Nebraska–Lincoln, Lincoln, USA

2 University of Minnesota–Twin Cities, Minneapolis, USA

3 Youngstown State University, Youngstown, USA

4 Universidad Nacional de Colombia, Bogota, Colombia