The Effect Of Comments On Program Comprehension: An Eye-Tracking Study

Youssef Abdelsalam · Norman Peitek · Annabelle Bergum · Sven Apel

Received: 13.01.2025 / Accepted: 19.08.2025

Abstract Programmers rely on code documentation and comments to understand source code, with program comprehension tasks consuming a significant portion of development time. Despite their importance, the impact of comments on program comprehension remains debated. Our study addresses this gap by investigating the influence of comments on program comprehension.

Employing a mixed-methods approach, we conducted an eye-tracking study involving 20 computer science students to explore the influence of code comments on program comprehension. By analyzing both quantitative and qualitative data, we aim at comprehensively assessing the influence of comments on various aspects of program comprehension. The quantitative data collected consists of behavioral metrics assessing program comprehension in terms of correctness and response time, along with gaze data providing insights into visual attention, linearity of reading order, and gaze strategies. This was complemented by the participants' ratings on the perceived difficulty and contribution of comments. Additionally, we gathered participants' experiences through a post-questionnaire, enriching the analysis with qualitative insights into the effectiveness of comments, navigation strategies, and overall experiences with comments.

Our findings reveal that the effect of comments on supporting program comprehension varies significantly across code snippets, ranging from a 30% decrease to a 34% increase in performance. Comments significantly guide visual attention, accounting for up to 23% of fixations, and promoted a more linear reading approach. Participants predominantly adhered to a "code-first" strategy. Moreover, comments were rated positively for clarifying complex segments of code and contributing to program comprehension. However, this favorable perception did not consistently translate into improved performance or reduced perceived difficulty across snippets.

This work has been supported by ERC Advanced Grant 101052182 as well as DFG Grant 389792660 as part of TRR 248 – CPEC.

Youssef Abdelsalam
Saarland University, Germany
E-mail: abdelsalam@cs.uni-saarland.de

Based on our findings, we propose avenues for future research, including comparative studies on automated versus human-generated comments and the development of predictive models for assessing comment usefulness.

Keywords Software Engineering · Program Comprehension · Eye Tracking · Comments

1 Introduction

Programmers spend a considerable amount of time reading and comprehending source code. According to Dunsmore et al. [29], program comprehension tasks have been reported to take up as much as 60% of the total development and maintenance time. During that process, programmers frequently rely on code documentation and comments, which inform about the code by clarifying its purpose, functionality, and design reasons.

Research suggests that comments facilitate an eased understanding of complex code while decreasing programmers' cognitive load [57]. They can play a crucial role in maintaining code quality and readability, and are also suggested to serve as "beacons" (i. e., guiding signals highlighting and drawing attention to specific elements [19]) that aid in comprehension and analysis of complex code. However, the way and extent to which comments influence program comprehension is still not well understood. On the other hand, comments can also be a source of distraction, especially if they are excessive, irrelevant, or obsolete. Previous studies have found that extensive commenting can negatively affect program comprehension, leading to longer reading times and potentially reducing the programmer's productivity [65].

Despite the importance of documentation in software development [96], limited empirical research has been conducted to establish *how* programmers utilize code documentation, especially comments, for program comprehension. Analyzing how programmers commonly approach information exploration during software development and maintenance is needed, if such comprehension is to be facilitated by tool developments or adjustments to the software development process (e. g., guidelines on writing comments).

Our study involved an eye-tracking experiment with 20 computer science students to empirically explore the influence of code comments on program comprehension. Participants were presented with 12 Java code snippets, both with and without comments, while their eye movements were recorded. We collected multiple types of quantitative data: performance metrics (correctness and completion time), eye-tracking metrics (fixations, durations, and reading patterns), and Likert-scale ratings from a post-questionnaire. Additionally, the post-questionnaire included open-ended questions regarding each aspect to gather qualitative insights into participants' experiences and strategies. Through this mixed-methods approach, we aim to provide valuable insights into the role of comments in shaping cognitive processes during program comprehension, combining objective measurements of performance and visual attention with participants' self-reported opinions.

To guide our investigation, we address the following research questions:
RQ1: *How do comments affect program comprehension performance?*
In particular, we examine the influence of comments on correctness and response time.
RQ2: *How do comments affect reading behavior during program comprehension?*
This is explored through three sub-questions:

RQ2.1: Influence on visual attention allocation
We examine this through analyzing fixation counts and durations.
RQ2.2: Influence on linearity of reading order
We study this by comparing sequential and non-sequential reading patterns.
RQ2.3: Influence on gaze strategies
We examine if participants adopt a “code-first” or “comment-first” strategy.

RQ3: *How do participants perceive the role of comments in facilitating program comprehension?*
Specifically, we analyze Likert-scale ratings on task difficulty and comment helpfulness.

Our findings reveal that the effect of comments on supporting program comprehension varies significantly across different code snippets, with effects on program-comprehension performance measures ranging from a decrease of 30% to an increase of 34%. We found that comments significantly guide visual attention, accounting for up to 23% of all fixations, and promote a more linear reading approach within code lines. Participants predominantly adhered to a “code-first” strategy, prioritizing code before considering comments. Moreover, comments were rated positively for clarifying complex segments of code and contributing to program comprehension. However, this favorable perception did not consistently translate into improved performance or reduced perceived difficulty across snippets. This discrepancy between perceived and actual contribution highlights the necessity of prioritizing quantitative metrics over subjective viewpoints when considering strategic commenting practices.

Additionally, our thematic analysis provides deeper insights into how comments influence programmers’ comprehension strategies. Comments are highly valued for providing structure and summarizing complex code segments, which helps in making the code easier to understand. They also offer necessary context, facilitating comprehension, especially in ambiguous or complex scenarios. However, the effectiveness of comments is closely tied to their relevance and quality, with misleading or redundant comments being criticized. The analysis also highlights that programmers often adopt a selective approach to reading comments, based on their understanding of the code, and that comments can significantly alter reading strategies, shifting from a purely top-down approach to a more targeted engagement with specific lines of interest. Despite the variability in reliance on comments, they are consistently regarded as essential for navigating and understanding challenging code blocks, though their mere presence can sometimes increase the perceived complexity of the code.

The findings from this study have several practical implications. First, commenting guidelines should emphasize quality and contextual relevance rather than quantity, as comments are not universally beneficial and may hinder comprehension if poorly written. Additionally, since comments can influence attention and reading patterns, this suggests opportunities for IDEs and educational tools to support natural reading behaviors. Furthermore, understanding individual gaze strategies opens the door to recommending code and comment structures that align with diverse reading approaches. Another important implication is the discrepancy between perceived and actual helpfulness of comments, which highlights the need for developer training on effective commenting, critical reading, and evaluative practices. Finally, programming tools that assist in writing, reviewing, or filtering comments can help developers reduce cognitive load and focus on the most relevant parts of the code. These implications inform programming education, development of coding standards, and the design of tools to support program comprehension, ultimately contributing to more efficient software development and maintenance practices.

In summary, we make the following contributions:

- An eye-tracking experiment to investigate the impact of comments on program comprehension performance, visual attention, linearity of reading order, gaze strategy, and participants' self-reported opinions.
- Empirical evidence on the diverse impact of inline comments on program comprehension across different code snippets.
- Empirical evidence that comments significantly guide visual attention and promote a more linear reading approach within code lines.
- Increased external validity by using a modern programming language (Java) and more realistic comprehension tasks, improving the generalizability of our findings to contemporary software development.
- A thematic analysis of participant responses on the role of comments in program comprehension, revealing key themes in how comments are perceived and utilized.
- An online replication package[1], including experiment design, raw data, and executable analysis scripts.

[1] `https://github.com/brains-on-code/eyetracking-program-comprehension-comments`

2 Background

In this section, we provide fundamental contextual and background information required to thoroughly understand the significance of our proposed study on the effect of comments on program comprehension performance, visual attention, linearity of reading order, and gaze strategy. Following a discussion of program comprehension, we examine some of the literature on comments in software development. Furthermore, we present a review on the incorporation of eye-tracking technology in research related to software development.

2.1 Program Comprehension

Software development and maintenance rely on program comprehension [50]. The process of program comprehension describes the internal cognitive processes that occur when programmers are engaged in reading and comprehending source code. This involves the understanding and interpretation of high-level source code.

Programmers use a variety of cognitive strategies to comprehend programs, depending on factors related to (1) the programmer, (2) the program, and (3) the task [85]. Programmer factors include the skills, knowledge, and familiarity of the programmer with the program and application domain. Program factors include the program's domain, size, complexity, quality, and documentation availability. Task factors include the type, size, complexity, time constraints, and environment of the comprehension task. Each of these aspects plays a role in deciding which comprehension method to choose, and can change from one context to the next.

However, there are challenges associated with program comprehension. Programmers frequently struggle with linking different conceptual areas, such as connecting the problem or application domain to the solution, navigating different levels of abstraction, reconciling the system's design or description with its actual implementation, and reconciling the associative nature of human cognition with the formal world of software [81].

Due to these difficulties, substantial research has been conducted to offer explanations and insights into the cognitive processes underlying program comprehension, establishing it as a distinct area of research within software engineering. Central to this line of research is the concept of cognitive load, which refers to the mental effort required to process and comprehend information.

In the following sections, we summarize relevant research associated with program comprehension, including cognitive load, mental models, knowledge domains, and cognitive models.

2.1.1 Cognitive Load Theory

Cognitive load theory (CLT), introduced by Sweller [110], provides a foundational framework for understanding the mental effort required during complex tasks. It distinguishes between two forms of memory: *long-term memory*, which stores information indefinitely, and *working memory*, which temporarily holds and processes information relevant to ongoing cognitive activities [5, 128]. The theory postulates that the human working memory is limited in capacity and can only hold and process a fixed number of items ("chunks") at a time [23, 63, 69]. When this capacity is exceeded, individuals may experience cognitive overload, resulting in impaired task performance and increased error likelihood [23, 69].

CLT defines *cognitive load* as a multi-dimensional construct representing the demands placed on working memory during task execution. In the practice of software engineering, these demands arise, for example, when developers engage with complex codebases or make intricate code changes. Under such circumstances, cognitive overload may impair their ability to complete tasks efficiently and accurately [118].

CLT further differentiates between two key types of cognitive load: *intrinsic* and *extraneous*. Intrinsic load is determined by the inherent complexity of the task itself, for example, the interdependencies among components in a software system or the conceptual difficulty of a required code change [23, 110]. In contrast, extraneous load arises from the way information is presented or structured. Poor documentation, convoluted syntax, or unstructured code representations can unnecessarily increase the cognitive effort required to perform a task [3, 17].

To assess cognitive load, researchers employ a range of measurement techniques, including subjective, performance-based, and physiological methods [23, 69, 105]. Subjective measures, such as the NASA-TLX questionnaire [40], capture participants' self-reported workload. Performance measures assess the effects of cognitive load through indicators such as response time and task accuracy. Physiological measures offer insights into cognitive load and mental effort, drawing on eye-tracking metrics—such as fixation count, fixation duration, and pupil dilation—as well as neural signals like electroencephalography (EEG) [105].

In this work, we focus specifically on performance-based indicators and eye-tracking metrics (fixation counts and durations) as proxies for cognitive load during program comprehension tasks.

2.1.2 Mental Models

The notion of the mental model in program comprehension was initially put forward by Storey et al. [109]. They defined it as the programmer's mental representation of the program. This representation is constructed through observation, inference, or interaction with the program [85]. The accuracy and comprehensiveness of a mental model's development can fluctuate based on variables such as the programmer's level of expertise and the complexity of the program. Von Mayrhauser and Vans [60] proposed that mental models typically include both static and dynamic components, which serve as a conceptual framework for organizing and interpreting program information. These models are further shaped by cognitive enablers, such as strategies and techniques that further support program comprehension.

Static Elements Although the structure of a mental model may differ among individual programmers, certain entities always appear in each mental model. These entities, referred to as static elements, include text structures [11], chunks [25, 53], plans [60], and hypotheses [54].

Dynamic Elements The mental model's dynamic elements are linked to the strategies used by programmers to make sense of the code. Strategies refer to the systematic sequence of actions and procedures that are employed to reach the desired level of comprehension. Littman et al. [56] distinguish between two broad categories of strategies: systematic and as-needed. In systematic approaches, programmers strive to comprehend the entire program prior to performing development or maintenance tasks. As-needed strategies, focus on comprehending only the specific program parts that need maintenance. A second classification of strategies includes shallow and deep reasoning. In shallow reasoning, important lines of code are rapidly scanned and beacons are utilized to identify mental model plans. Deep reasoning requires a comprehensive program synthesis and the development of causal links between functions and objects. This method is generally used when the programmer has little to no experience with the program.

Cognitive Enablers Program elements that aid in comprehension are known as cognitive enablers. Beacons and rules of discourse are two types of cognitive enablers. Beacons [19, 122] are indicative markers embedded within a program that serve as cues for specific structures or operations. They might be useful for finding code patterns or specific functions that developers need. Rules of discourse, as defined by Soloway and Ehrlich [103], specify programming conventions such as standard algorithms, data structure implementations, and coding standards, which direct programmers in their interpretation and comprehension of the program.

2.1.3 Knowledge Domains

The comprehension of programs is significantly influenced by knowledge domains, which function as fundamental units for comprehending programs. A knowledge domain is a finite collection of primitive objects, the interactions between them, and the operators that can change those relationships or attributes [19]. Program comprehension requires bridging and mapping between different knowledge domains, especially between the problem domain (the real-world problem the program is intended to solve) and the program domain [121].

The problem domain, also known as the application domain [18, 87], refers to the specific part of the real world that a program aims to address and solve problems within [82]. It is characterized by a shared vocabulary, common assumptions, solution approaches, and an existing body of literature independent of the programs that tackle these problems.

The program domain focuses on the knowledge domain associated with programming and the source code level [18]. It encompasses both general programming concepts and language-specific concepts, reflecting the chosen programming paradigm (object-oriented, imperative, functional, etc.).

2.1.4 Cognitive Models

Cognitive models are vital for understanding program comprehension. While mental models focus on the programmer's representation of the program, cognitive models explore broader cognitive aspects of software comprehension. These models provide insights into how programmers acquire, organize, and apply knowledge about programs, highlighting underlying thought processes. Key established models in this field are summarized in Table 1.

Table 1 Summary of Cognitive Models in Program Comprehension

Model	Year	Description
Shneiderman and Mayer [97]	1979	Views program comprehension as a bottom-up task, starting with individual code statements grouped into higher-level structures, relying on syntactic and semantic knowledge.
Brooks [19]	1983	Involves mapping between the problem domain and the program domain, emphasizing iterative construction of knowledge through a top-down approach. Expertise in the problem domain helps define initial hypotheses, refined using code beacons.
Soloway and Ehrlich [103]	1984	Highlights the role of plans and rules of discourse. Expert programmers, familiar with these rules, comprehend faster unless the program violates expected discourse rules.
Pennington [75]	1987	A bottom-up approach that builds a program model from control-flow representation and a situation model that maps high-level domain plans, creating an abstract understanding of the program's goals.
Letovsky [54]	1987	Comprehension involves a knowledge base, a mental model, and an assimilation process (either top-down or bottom-up) based on the programmer's current knowledge source.
Soloway, Adelson, and Ehrlich [102]	1988	Views comprehension as a top-down task involving external and internal representations, and plans at strategic, tactical, and implementation levels, mapped to the program code.
von Mayrhauser and Vans [59]	1994	Combines top-down and bottom-up approaches, suggesting that familiarity with the program dictates the approach. Recognized programs trigger a top-down strategy, while unfamiliar code requires building a control-flow abstraction.

2.2 Comments in Software Development

The comprehension of source code holds significant importance for programmers, as highlighted by Knuth [49], Standish [108], Tiarks [116], and Siegmund [98]. The inclusion of comments in code has been suggested to enhance the comprehension process, as shown by studies conducted by Elshoff and Marcotty [31] and Corazza et al. [24]. Nevertheless, comments frequently prove to be inadequate in practice: It has been observed that leaving new code without comments and neglecting to update existing comments in tandem with code modifications is prevalent [35, 44, 104]. To tackle this matter, scholars have put forth methodologies for identifying outdated comments, as seen by the works of Tan et al. [113], Sridhara [106], and Ratol [78]. However, the impact of comments on program comprehension remains uncertain and therefore requires empirical scrutiny. Some argue for self-explanatory code, emphasizing the creation of code that is easily understandable without relying heavily on comments [48]. In this section, we provide a brief exploration of comment characteristics as fundamental elements of source code documentation.

2.2.1 Types and Structure of Comments

Source code comments can be classified into two types: inline comments, which occupy a single line, and block comments, which can span multiple lines. Programming languages may support one or both types [119]. The structure of comments comprises three elements: comment extent, comment target, and comment category [96].

While a single explanation can span multiple comment tags, the extent is defined as a sequence of consecutive tags forming a continuous text, representing the comment as a whole. Comments target specific elements within the code, and these targets can be categorized relative to surrounding syntax elements into four types: Left, Right, Parent, and In-Place, as shown in Table 2 [96]. Furthermore, researchers classified comments based on their content and relationship between the comment and its target, identifying 11 categories, as detailed in Table 3 [70,96].

Table 2 Types of comment targets. [96]

Target	**Description**	**Example**
Left	Targets the syntax element that ends immediately before the comment.	`thread.join(); // Let the job finish.`

Right	Targets the syntax element that starts immediately after the comment.	`// Copy the array.`
		`for (int i = 0; i < a.length; i++) {`
		`b[i] = a[i];`
		`}`

Parent	Targets the parent element containing the comment, commonly seen in constructs such as loops or conditional blocks.	`if (obj == null) { // error`
		`return;`
		`}`

In-Place	The comment itself is the target, such as metadata or author notes.	`// Author: John Doe`

Table 3 Types of comment categories.

Category	**Description**
Postcondition	Conditions that hold after execution explaining "what" the code does.
Precondition	Conditions that hold before execution, including statements that hold regardless of code execution, explaining "why" the code is needed.
Value Description	Phrases that can be equated with variables, constants, or expressions.
Instruction	Instructions for programmers, often denoted as `TODO` comments.
Guide	Guides for code users, distinct from instructions.
Interface	Descriptions of functions, types, classes, or interfaces.
Meta Information	Meta information such as author, date, or copyright.
Comment Out	Code that has been commented out, lacking a specific target.
Directive	Compiler directives not intended for human readers.
Visual Cue	Text inserted for ease of reading, e. g., indentation or section headers.
Uncategorized	All other comments that do not fit into the above categories.

2.2.2 Comment Content and Practices

The content of comments in source code has been extensively debated within the software development community. A good comment should be consistent with the code it describes, providing clarity and avoiding misleading information [112].

Comments use a specialized sublanguage characterized by limited vocabulary, specific syntactic and semantic constructions, present tense, indicative or imperative mood, and a limited set of verbs [32]. Studies have shown that comments often contain problem domain terms, with about 23% of such terms appearing in comments alone compared to 11% in identifiers [39].

Maintaining and updating comments as source code evolves can be challenging. Research indicates that comments and code rarely co-evolve, with changes in comments typically triggered by changes in the corresponding source code [35]. However, the percentage of commented functions has been found to remain consistent over time [44].

Comment density, the ratio of comment lines to total lines of code, varies widely across software projects, from less than 1% to over 50% [34]. Practices for comment usage are influenced by coding style guides and conventions, which provide recommendations to enhance code readability and maintainability. Examples include the Google Java Style Guide[2] and the Oracle Java Code Conventions[3].

Overall, the content, language, size, and practices related to comments play a crucial role in program comprehension and maintainability. While opinions vary on the ideal amount and style of commenting, well-written and informative comments are generally agreed to enhance the readability and comprehensibility of source code.

2.3 Eye Tracking in Software Engineering Research

Eye tracking is a method for gathering information about a participant's visual attention by observing eye gaze patterns [28, 79]. Visual attention is essential to the cognitive processes of understanding and problem solving in programming, and these same cognitive processes direct visual attention to particular regions. As a result, eye tracking is useful in analyzing the cognitive processes and effort of programmers during software engineering activities [28]. By understanding how eye tracking provides insights into cognitive processes, we can explore its applications in software engineering research and how it contributes to understanding various aspects of software development, such as source code reading, debugging, comprehension of software artifacts, and software traceability [91].

In the field of software engineering, eye tracking finds application in tasks such as source code reading, debugging, comprehension of software artifacts, and software traceability [91]. Recent studies have also combined eye tracking with other neuroimaging and biometric techniques, such as EEG, functional magnetic resonance imaging (fMRI), and functional near-infrared spectroscopy (fNIRS), to assess task difficulty and cognitive burden [33,36,52,74]. However, the wide variety of eye-tracking devices, techniques, metrics, and analyses employed by researchers

[2] `https://google.github.io/styleguide/javaguide.html`

[3] `https://www.oracle.com/java/technologies/javase/codeconventions-contents.html`

presents a major challenge, as it complicates the comparability and replication of study approaches.

In what follows, we introduce the foundations of eye tracking, essential for understanding its insights into participants' cognitive processes during program comprehension tasks. We will explore the types of data collected by eye trackers and how these data are aggregated and utilized in research.

2.3.1 Foundations of Eye Tracking

The analysis of eye-tracking data presents some challenges, especially with regard to program comprehension [7,42]. In this section, we provide definitions and metrics for analyzing eye-tracking data in software engineering research.

First-Order Data The term "first-order data" refers to the raw and unprocessed information obtained from eye-tracking devices [42]. One of the primary metrics used is the X and Y positions, which refer to the spatial coordinates of each gaze point, providing insight into the participants' focal point of attention on the stimulus.

Several variables can impact the accuracy and reliability of this metric, including ambient light levels, the distance between the eye tracker and the individual, and the quality of the camera image [100].

Second-Order Data Second-order data in eye tracking are computed from raw, first-order gaze data by applying physiological thresholds. This transformation relies on event detection algorithms that apply spatial and temporal criteria to segment raw eye gaze data into distinct events [84]. However, it is important to note that the selection of event detection algorithms can affect the outcomes of data analysis. The most commonly analyzed second-order events are:

– Fixations: A spatially-stable eye gaze typically lasting 100–300 milliseconds, focusing on a specific area and initiating cognitive processes [46]. The duration varies with task and participant characteristics.
– Saccades: Rapid, continuous eye movements between fixations, typically lasting 40–50 milliseconds, allowing limited visual perception.

According to psychological research, the acquisition and processing of information predominantly take place during fixations. It has been observed that participants can effectively comprehend a complex visual stimulus with only a limited number of fixations [77].

Context is crucial when attempting to make sense of fixations. A higher fixation rate on a specific area of interest (AOI) may suggest an increased degree of engagement with its content. Nevertheless, it is worth noting that an accumulation of fixations may also indicate increased effort or difficulty with comprehending the stimulus [76].

Fixations may also be either voluntary or involuntary. While involuntary fixations are caused by reflexes, such as the optokinetic reflex, which focuses attention on moving things, voluntary fixations are the deliberate concentration on certain elements. Within the field of software engineering, the primary area of interest for researchers lies in the examination of voluntary fixations. [90].

Therefore, it is imperative to perform a thorough cleaning of the data prior to the analysis due to the presence of various sources of interference, such as noise, outliers, and invalid entries [100]. Researchers may either visually clean the data by reviewing fixations and saccades and erasing plainly wrong entries, or statistically clean the data by deleting outliers and abnormally extended fixations.

Third-Order Data By analyzing fixations and saccades, the following third-order metrics can be derived:

- Fixation count: The number of fixations that occurred within a specific AOI or the entire stimulus.
- Fixation duration: Also referred to as fixation time, denotes the aggregate duration including all fixations made on a specific AOI or the stimulus under examination.
- Percentage of fixations or fixation rate: Refers to the proportion of total fixations on a AOI or stimulus in relation to another.

Prior research in the field of eye tracking has employed various metrics such as fixation count, fixation duration, and fixation rate to identify AOIs that cause more attention from individuals [26,27,117]. These metrics have been utilized to evaluate the efficacy of participants' problem-solving approaches [101]. A lower fixation rate suggests decreased efficacy in search tasks, indicating that participants exert more effort to identify relevant regions [76]. On the contrary, elevated rates of fixation serve as an indication of higher effort necessary for completing different tasks, including finding bugs [8, 92], debugging [93], comprehending source code [13], recalling identifier names [94], or examining various stimulus layouts [38, 127].

To make appropriate comparisons between two AOIs or stimuli, it is essential to modify the values based on their proportions. In the context of textual analysis, it is necessary to normalize fixation counts by dividing it by the word count of each AOI. This normalization technique allows for fair and accurate comparisons between AOIs that may have varying word counts [90].

The third-order metrics provided by saccades are comparable to those provided by fixations and include:

- Saccade count: Total number of saccades within an AOI or in response to the stimulus.
- Saccade duration or saccade time: the duration of all saccades within an AOI or the stimulus.
- Regression rate: the proportion of backward or regressive saccades relative to the total number of saccades (e. g., leftward in left-to-right source code reading) [22, 76].

Research suggests that increased regression rates correspond to an increase in the difficulty of executing and finishing a task [37, 76]. Source code reading has been shown to have greater regression rates than natural language text [22]. Saccades were also used by Fritz et al. [36] to study how stimulus difficulty affected participants.

Fourth-Order Data Scan paths refer to sequences of saccadic movements and fixations across a stimulus, forming a linear chronological sequence of fixations or visited AOIs. The analysis of scan paths provides valuable insights into the temporal and spatial characteristics of fixations, thereby serving as reliable indicators of search efficiency. Longer scan paths show that participants spend more time and effort studying a stimulus to find relevant AOIs, suggesting less efficient scanning and searching.

Longer experiment sessions result in longer scan paths, which are more difficult to evaluate and compare. Number, position, timing, and length of fixations are all factors that must be taken into account.

One relevant method for analyzing scan paths is *linearity of reading order*. The concept of linearity is closely linked to the search strategies employed by participants [76]. In this context, linearity of reading order refers to the eye gaze patterns observed during reading, specifically the tendency to move from left to right and top to bottom, which is commonly observed among readers of Latin-based natural languages.

Spatial distribution-based metrics, including linearity of reading order, are susceptible to invalid data. For instance, a slight shift in the position of a single fixation may significantly impact the analysis. As a result, when using fourth-order data, noise reduction and data cleaning are essential [89].

2.3.2 Eye Tracking Assumptions and Limitations

The correlation between eye gaze and cognitive processing depends on two fundamental assumptions originating from the theory of reading: the immediacy assumption and the eye-mind assumption [46]. The immediacy assumption asserts that as soon as participants encounter a stimulus, such as when a reader reads a word, the interpretation of the stimulus starts immediately. The eye-mind assumption postulates that participants focus their attention solely on the stimulus element that is currently being processed.

These two assumptions serve as the foundation for the representation of participants' cognitive processes through eye gaze data. The analysis of eye gaze data provides valuable insights into participants' focus, cognitive load, and temporal dynamics involved in processing a given stimulus. Furthermore, based on physiological investigations, psychologists hypothesize that individuals do not have conscious control over many features of their eye gaze, such as pupil size and eye blinks, except for the position of their focus.

Eye Tracking Limitations Eye trackers have several inherent limitations that can impact the accuracy and reliability of gaze data [41,90]. Over time, calibration drift can reduce measurement accuracy due to physiological changes such as moisture fluctuations [64]. Furthermore, eye trackers capture only foveal vision–the direct line of sight–while peripheral or extrafoveal vision, which accounts for about 98% of the visual field, remains unrecorded [90]. Although technological advancements are gradually mitigating these issues, these limitations must be acknowledged when interpreting eye-tracking results.

2.3.3 Visualization of Eye-Tracking Data

Among the various visualization techniques available for eye-tracking data, the Radial Transition Graph Comparison Tool (`RTGCT`) [14] is a notable tool. `RTGCT` specializes in visualizing transitions between AOIs through radial transition graphs. These graphs visually depict how participants move their gaze between different AOIs, with sectors representing the duration of fixations and arcs showing the frequency of transitions between AOIs. The thickness of an arc indicates the number of transitions between two AOIs, while small circles within the AOI sectors mark the direction of these transitions–black for outgoing and white for incoming. Figure 1 shows an example radial transition graph generated by the `RTGCT`.

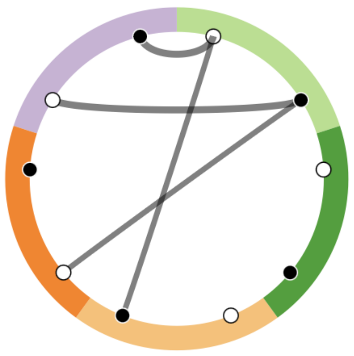


Fig. 1 Example of radial transition graph generated by the `RTGCT` [14].

`RTGCT`'s interface aids further analysis by enabling the comparison of multiple participants' data through diff graphs, which highlight differences in fixation durations and transition patterns between two datasets. Additionally, `RTGCT` offers a comparison grid view, where multiple radial transition graphs can be compared side by side, aiding in the identification of similarities and differences in eye movement patterns across various participants and stimuli [14].

3 Related Work

In this section, we review the existing literature on the role of code comments in program comprehension. We summarize key studies that have explored how comments affect readability, comprehension, and developers' cognitive load. Additionally, we highlight the importance of understanding visual attention and reading patterns through eye-tracking data. This overview sets the foundation for identifying gaps in the current research and justifying the need for our study.

3.1 Existing Studies

3.1.1 Comments in Source Code

The role of comments in program comprehension has been studied extensively over the years through foundational theories and numerous empirical studies. Early foundational work emphasized the cognitive role of comments in bridging domains. Kernighan and Plauger (1974) [48] suggested that the best documentation includes "enlightening" comments. Brooks (1978) [18] emphasized comments as a bridge between the program domain and the problem domain, advocating for comments that facilitate this connection.

Empirical investigations into the effect of comments began in the early 1980s. Woodfield et al. (1981) [125] conducted an early experiment to examine the impact of comments and modularization on program comprehension. Experienced programmers were provided with different versions of a Fortran program, and the group given commented versions performed significantly better in answering comprehension questions. Tenny (1985) [114] extended this line of research studying the readability of the Banker's Algorithm, again finding that comments improved readability.

Nurvitadhi et al. (2003) [67] conducted an experiment comparing the usefulness of class-level and method-level comments. Their findings indicated that participants who received method-level comments performed better on comprehension quizzes than those who received class-level comments, reinforcing earlier results on the value of code comments.

As the field matured, researchers began to investigate how developers use comments in practice. Ying et al. (2005) [126] found that developers commonly use comments for internal communication, such as `TODO` notes, suggesting that comments also support coordination and planning tasks within development teams.

Another line of work has focused on the readability and quality of comments. Buse and Weimer (2010) [20] developed a metric to assess the readability of comments, showing a strong correlation between their readability scores and software quality indicators such as defect rates and code changes.

With the rise of automatic comment generation, several works explored how to create more helpful documentation. Sridhara et al. (2010) [107] proposed a technique for automatically generating summary comments for Java methods to keep documentation up-to-date. Wong et al. (2013) [124] and McBurney and McMillan (2014) [62] proposed techniques for generating helpful comments to support comprehension. In a related study, McBurney and McMillan (2014) [61] determine the characteristics of "good" comments and found that effective author-written comments often include keywords from the source code.

The role of comments in requirements traceability has also received attention in the past. Antoniol et al. (2002) [4] described a method for automatically recovering traceability links between code and documentation by analyzing identifier names. Ali et al. (2014) [2] conducted an empirical study on the influence of comments in traceability tasks. They found that programmers spent an average of 4.5 seconds reading comments and ranked them as the second most important feature after method names. Building on this, Ji et al. (2015) [43], Seiler and Paech (2017) [86], and Krüger et al. (2018) [51] proposed integrating comment-like annotations in the source code to support feature traceability and found them beneficial in practice.

Nielebock et al.(2018) [65] investigated the effectiveness of comments in program comprehension for small programming tasks. They conducted an experiment involving 277 participants, mainly professional software developers, who performed programming tasks on differently commented code. The study aimed to replicate previous findings, examine the performance of participants with varying levels of experience, and explore developers' opinions on comments. The results indicate that comments were considered less important for small programming tasks compared to other mechanisms such as proper identifiers. However, participants acknowledged the necessity of comments in specific situations. The study adds to the existing body of research on the uncertain impact of comments on software development.

3.1.2 Eye Tracking and Program Comprehension

In an exploratory study by Parkin (2004) [71], experienced C programmers performed maintenance tasks on a C program. Contrary to previous research, programmers implementing corrections relied more on program documentation and header information than those working on enhancements. Enhancers made specific use of task documentation to map out extensions and verify code modifications. This study sheds light on the program comprehension strategies employed during different maintenance tasks.

Blinman and Cockburn (2005) [15] focused on the effects of naming style and documentation on the comprehensibility of source code. They examined software development frameworks and their impact on programmers' usage at the source code level. They found that using a descriptive interface naming style positively influences program comprehension. Additionally, documentation is shown to be important but increases the time spent studying the source code.

Busjahn et al. (2015) [21] investigated the linearity of reading source code compared to natural language text. They found that novices read source code less linearly than natural language text, and experts read code even less linearly than novices. These results highlight the specific differences in reading approaches between source code and natural language text, suggesting that non-linear reading skills increase with expertise.

Shinyama et al. (2018) [96] developed a method to identify explanatory code comments that enhance program comprehension. They proposed eleven categories of code comments and used a decision-tree classifier to achieve 60% precision and 80% recall. The researchers analyzed 2 000 GitHub projects and found two dominant comment types: preconditional and postconditional. Their findings also revealed consistent grammatical structures in English code comments across different projects.

Adeli et al. (2020) [1] explored how providing the right information at the right time and place enhances program comprehension. Using a dedicated IDE, annotations were implemented to facilitate access to relevant information. A user study with 22 novices showed that this approach improved accuracy and reduced cognitive load during program comprehension tasks without compromising tool usability.

Beelders (2022) [9] revealed interesting insights into the reading patterns of novices and experts when it comes to source code: Novices tend to read method signatures with less attention compared to code lines within the method body.

They also show a tendency to transition between all lines of code without recognizing the relevance of each line. In contrast, experts demonstrate the ability to disregard irrelevant lines and focus on the current line being read. Furthermore, novices have a tendency to repeatedly read each line within a loop, while experts concentrate on lines with higher computational complexity in successive iterations.

Sharafi et al. (2022) [88] investigated the code navigation strategies of developers during realistic bug-fixing tasks using eye-tracking and IDE interaction data. Their study involved 36 participants working on Java bug fixes in Eclipse, combining gaze metrics with edit and scrolling behaviors to analyze attention distribution and transitions between code elements. The authors proposed a three-phase model—finding, learning, and editing—to characterize navigation behavior, validated by significant variations in fixation counts, durations, and saccadic lengths across phases. Key findings are that developers dynamically shift focus between code elements (e.g., method bodies, comments) as tasks progress, with excessive switching ("thrashing") correlated with lower effectiveness. Unlike prior studies limited to small code snippets, this work emphasized ecological validity by allowing full IDE interactions.

Karas et al. (2024) [47] conducted an eye-tracking study investigating programmers' visual attention during reading and writing tasks involving Java code summaries. Based on abstract syntax trees, they mapped gaze data onto fine-grained syntactic elements to analyze how cognitive load and attention vary during program comprehension activities. Their results showed that participants allocated the most visual attention to variable and method declarations, conditional statements, parameters, and method calls in both reading and writing tasks. Notably, comments received relatively low visual attention during reading but gained increased attention during writing tasks, suggesting that comments play a more active role when programmers generate or revise code summaries rather than when passively reading code.

Wallace et al. (2025) [120] conducted a comprehensive eye-tracking study to investigate visual attention during code summarization at the project level. The study involved 10 Java programmers, each participating in five one-hour sessions. During these sessions, participants were tasked with writing structured summaries that described the purpose of individual methods, their specific functionalities, and their relevance within the broader context of the project. Eye-tracking data were collected while participants had full access to entire codebases stripped of documentation. Findings include that programmers spent 40-65% of their fixation time examining class and instance methods within the same class as the target method, with significantly less attention devoted to methods in the call graph despite explicit instructions to include contextual information. As participants progressed through sessions, they had 35% fewer fixations and 13% lower regression rates while maintaining summary quality, indicating efficient knowledge retention and reduced cognitive load.

3.2 Rationale for our Study

The existing experiments and discussions in the literature support the notion that comments play a crucial role in program comprehension [15, 65, 67, 114, 115, 125]. They aid programmers in understanding code by providing additional explanations, clarifications, and insights into the program's functionality, which can significantly improve readability and comprehension. However, despite the existing body of research, there are several reasons that justify the need for our study.

Addressing Research Gaps To address existing research gaps, our study aims to provide a more comprehensive understanding of how comments influence program comprehension. While existing studies examined the role of comments (cf. Section 3.1.1), no existing work has combined behavioral, visual, and qualitative data to capture their effects on program comprehension. To fill this gap, we investigate five distinct aspects of how comments influence program comprehension. First, we analyze behavioral data, including task correctness and response times, to evaluate the overall impact of comments on performance. Second, we employ eye tracking, which allows us to collect precise data on how programmers visually process and comprehend code. This approach offers valuable insights into the cognitive load involved in program comprehension and the role of comments in guiding attention and facilitating comprehension. Specifically, we analyze fixation patterns on areas of interest (code and comments) to determine how comments influence where and how long participants focus during program comprehension. Third, we examine the linearity of code reading order and investigate how the presence of comments affects the sequential and orderly reading of code elements. Fourth, we investigate the employed gaze strategies during program comprehension, specifically focusing on how participants transition their visual attention between code and comments. Finally, we evaluate participants' self-reported opinions and ratings of task difficulty and perceived helpfulness of comments. By integrating these perspectives, our study aims to provide a more thorough analysis of the effects of comments in program comprehension. This will contribute to the existing body of knowledge and offer valuable insights for software development practices.

Another critical aspect our study addresses is the isolation of the effect of comments. Previous studies have often considered additional aspects of programs, such as modularity [125] or identifier names [83], which makes isolating the effect of comments challenging. This poses a threat to the internal validity of those studies. Moreover, the measurements used in prior research are often subjective, resulting in limited quantitative results. The potential presence of confounding factors, which may affect program comprehension performance, visual attention, linearity, and gaze strategies, makes it difficult to analyze the isolated effect of comments. Therefore, our study fills this gap by conducting research that controls for potential confounding factors, providing a clearer understanding of their influence.

Enhancing External Validity Many previous studies were conducted with older programming languages and paradigms [30, 66, 95, 111, 114, 115, 125], which may compromise the applicability of their results in modern software development practices. To enhance the external validity and relevance of the findings, our study investigates the effects of comments on program comprehension using a contemporary programming language (Java). Consequently, we aim to provide insights directly applicable to present-day software development, improving the generalizability of the findings.

Replication and Consolidation Lastly, replication studies in empirical software engineering are crucial for validating and consolidating existing knowledge [6,12,45]. Therefore, we emphasize the importance of conducting further replications to gain a deeper understanding of the effects of comments on program comprehension. By replicating and consolidating previous findings, we can establish a more robust foundation of knowledge regarding the impact of comments, addressing varying and sometimes contradictory results from prior studies. Such replication efforts contribute significantly to the advancement of the field and help establish more reliable and generalizable conclusions about the role of comments in program comprehension and reading strategies.

4 Study Design

4.1 Aim

Our study aims at exploring the impact of comments on program comprehension performance, visual attention, linearity of reading order, gaze strategies, and participants' self-reported opinions on the role of comments in program comprehension using a combination of quantitative and qualitative methods. The following section provides an overview of the research questions, study design, variables and measures, code snippets, participant details, materials, procedures, and ethical considerations. By examining these aspects in detail, our study intends to enhance our understanding of how comments influence program comprehension.

4.2 Research Questions

> **RQ$_1$** How do comments affect program comprehension performance?

This research question addresses the impact of comments on program comprehension performance. It specifically examines how the presence or absence of comments influences participants' understanding and interpretation of the code snippets presented in the study. Program comprehension performance is operationalized through two objective behavioral metrics: task correctness and completion time (cf. Section 4.4).

> **RQ$_2$** How do comments affect reading behavior during program comprehension?

We have split our overarching second research question into three specific subquestions that we introduce next.

> **RQ$_{2.1}$** How do comments affect visual attention allocation during program comprehension?

This research question addresses the influence of comments on the allocation and distribution of visual attention during program comprehension. It investigates whether comments attract or redirect participants' gaze within the code snippets and how they impact participants' eye gaze patterns and fixations. In the context of our study, visual attention refers to participants' focus and allocation of attention while observing the code snippets. The effects of comments on visual attention were analyzed using eye-tracking data, providing insights into the role of comments in shaping participants' visual attention during program comprehension.

> **RQ$_{\mathbf{2.2}}$** How do comments affect the linearity of reading order?

This research question explores the impact of comments on the linearity of reading order. Linearity refers to the sequential and orderly reading of code elements and can play a significant role in program comprehension [21,73]. The question seeks to investigate how the presence of comments influences the flow and order of reading code snippets. By examining participants' reading patterns and comparing the linearity of reading order for snippets with and without comments, this research aims to gain insights into the effects of comments on the structured and sequential understanding of code. This investigation is intended as a non-exact replication of the studies by Busjahn et al. [21] and Peitek et al. [73], which have examined the role of reading order in program comprehension. However, our study aims to extend their findings by focusing specifically on the role of comments in influencing the reading order.

> **RQ$_{\mathbf{2.3}}$** How do comments affect participants' gaze strategies during program comprehension?

Gaze strategies refer to the intentional patterns and decisions participants make when directing their visual attention while observing code elements, including code and comments. In this study, we define and categorize gaze strategies into "code-first" and "comment-first" strategies based on participants' initial area of interest and their subsequent gaze transitions. The research question aims to explore how the presence of comments influences participants' gaze strategies, potentially affecting the prioritization and skipping of certain code elements. By comparing gaze strategies with and without comments, we seek to understand how comments guide attention, impact gaze transitions, and influence the cognitive processes involved in program comprehension.

> **RQ$_{\mathbf{3}}$** How do participants perceive the role of comments in facilitating program comprehension?

Understanding subjective self-reported ratings of code snippet difficulty and the role of comments in program comprehension is crucial to gaining a comprehensive understanding of the impact of comments. This research question investigates the subjective perspectives of participants on the difficulty of code snippets and the contribution of comments to program comprehension.

These five research questions will guide our study methodology and serve as the basis for analyzing the collected eye-tracking data. Further elaboration on the operationalization of these variables are provided in Section 4.4, where specific measures and methods for assessing program comprehension performance, visual attention, reading order, and participants' self-reported opinions are detailed. The research questions provide a clear framework for investigation, allowing for a systematic examination of the effects of comments on program comprehension. However, it is important to acknowledge that limitations or challenges may arise during our study, such as a small sample size or limited generalizability of the results. These potential limitations are discussed in Section 8.

4.3 Experiment Design

Our study employs a mixed-methods design, integrating quantitative and qualitative approaches to comprehensively investigate the impact of comments on program comprehension. The quantitative aspect involves presenting participants with Java code snippets in two conditions: Comments Missing (CM) and Comments Present (CP). We tracked the participants' eye movements using an eye tracker to capture their gaze data during comprehension, along with recording completion time and correctness. Additionally, participants provided Likert-scale ratings in a post-questionnaire, where they rated the difficulty of the code snippets and the helpfulness of the comments.

We adopt a within-subjects design, which allows for a comprehensive exploration of the effects of comments on each of the dependent variables. Each participant was presented with a set of Java code snippets, with each snippet representing a Java class that compiles and writes to the console. We carefully selected the code snippets to ensure a similar level of code complexity and comment relevance. Further details regarding the code snippet selection process and any necessary modifications are provided in Section 4.5.

We pseudo-randomized the snippet order to minimize order effects. We ensured that participants encountered a balanced mix of snippets with and without comments while also guaranteeing that each code snippet appears an equal number of times throughout the study. We also ensured that participants encountered a snippet only in one condition. For example, Participant 1 may encounter "Snippet 3 CP" (Comments Present) but not "Snippet 3 CM" (Comments Missing).

We made this design choice to mitigate potential carryover effects or learning biases that could occur if participants encounter the same code snippets in both conditions. However, it introduces additional challenges in the analysis, as the data points are not fully matched between conditions for each participant. The use of a linear mixed-effects model (LME) was appropriate for handling the within-subject variability in this partially crossover design. This model allows for the inclusion of both fixed effects (such as the presence or absence of comments) and random effects (to account for individual differences between participants) in the analysis (see Section 5.3).

The qualitative component of the study is also based on the post-questionnaire. The open-ended questions addressed each aspect of the study, providing insights into participants' experiences and strategies. These qualitative responses complement the quantitative findings by offering a deeper understanding of how comments influenced program comprehension, thus allowing for a more comprehensive analysis of the results.

4.4 Variables

Our study was structured around a single independent variable, namely the presence of comments within code snippets, which was manipulated across two distinct levels: Comments Missing (CM) and Comments Present (CP). This manipulation aims to explore how the inclusion or exclusion of comments affects program comprehension and navigation among participants. The impact of the independent variable on participants' program comprehension and navigation was assessed through five dependent variables, which are outlined as follows:

1. *Program comprehension performance*: This variable was operationalized by measuring participants' completion time and correctness for each code snippet. After reading a code snippet, participants were asked to write the output it produces. Their responses were scored based on the completion time and correctness of their answers, providing an indication of their level of program comprehension.
2. *Visual attention*: Visual attention refers to participants' focus and allocation of attention during program comprehension. The eye-tracking data provide insights into participants' fixation duration and frequency, indicating their visual attention on different code elements. We divided code snippets into code AOIs and comment AOIs, allowing for an analysis of the number and duration of fixations on specific elements, such as code lines and comments.
3. *Linearity of reading order*: Linearity refers to the sequential and orderly reading of code elements. The foundational work by Busjahn et al. [21] as well as subsequent research by Peitek et al. [73] provide many eye-gaze metrics to assess the linearity of reading order. These include local measures, such as vertical next text, vertical later text, horizontal later text, regression rate, line regression rate, and saccade length, as well as global measures such as the Needleman-Wunsch (N-W) Scores of the line reading order compared to the story and execution order. We used these measures to assess the linearity of the participants' reading order and the extent to which their gaze patterns align with the linear text reading order and the source code's execution order.
4. *Gaze Strategies*: Gaze strategies refer to the different approaches participants take when directing their gaze during the study. Two variations of gaze strategies were considered based on participants' initial area of interest and their subsequent gaze transitions: *code-first*, and *comment-first*.
   In the code-first strategy (cf. Figure 2), participants predominantly focus their gaze on the code elements of the given snippet, with the initial fixation often landing on a specific code line. Their visual attention is directed towards lines of code, variable names, and control structures, seeking to understand the logic and functionality of the program. Comments, if present, may receive occasional glances, but the primary emphasis is on the code itself.
   Contrary to the code-first strategy, the comment-first gaze strategy (cf. Figure 2) places a higher priority on comments rather than code lines. Participants' initial fixation often lands on a specific comment, seeking to gain context, explanations, and insights about the code's purpose and functionality.

Code elements are still observed but may receive less attention compared to the comments.

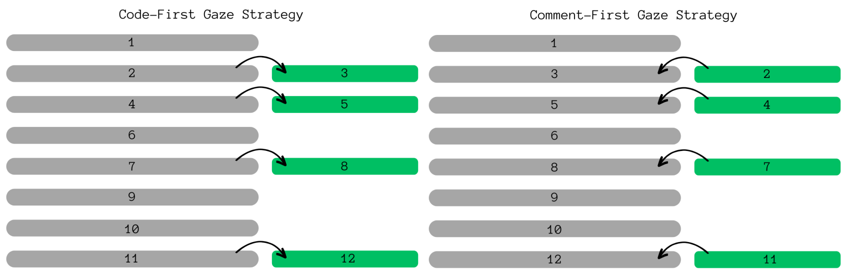


Fig. 2 Illustration of code-first and comment-first gaze strategies. Green boxes represent comment areas, and grey boxes represent code areas. The arrows indicate the direction of visual transitions between these regions.

To compare the measured reading order with the proposed reading orders, we used two types of measures: Locally, we compared code-to-comment and comment-to-code saccades to examine the order in which participants fixate on code elements and comments. Globally, we defined AOI sequences for each of the reading strategies and compared them to the actual reading order using the same N-W algorithm to determine the closest match.

5. *Participant's Self-Reported Opinions*: The subjective perception of each participant on the snippets and the role of comments in facilitating program comprehension was captured through evaluating their responses from the post-questionnaire as well as their Likert-scale ratings [55] of the snippet difficulty and comment contribution.

These measures for the dependent variables in our study have been previously validated in similar research studies and have demonstrated good reliability and validity [21, 72, 73]. Tasks assessing program comprehension have been widely employed in previous studies on program understanding and have been found to be reliable indicators of participants' comprehension levels. The eye-tracking measures, including fixation duration and frequency, have been extensively used in research on visual attention and reading patterns.

4.5 Code Snippets

Selection and Preprocessing We carefully selected the code snippets in our study to meet specific criteria concerning complexity and suitability for investigating the effect of comments on program comprehension performance, visual attention, linearity and gaze strategy. Instead of relying solely on one study as a source, we collected and curated a set of 12 appropriate code snippets from various studies [65, 72].

Our selection process involved evaluating the snippets based on their potential to accommodate meaningful comments. We considered both the presence of comments and their quality. Additionally, the code snippets should encompass programming concepts, such as recursion and dynamic programming, to further add complexity to the comprehension task.

To align the code snippets with the objectives of our study, we made several adaptations, including standardizing the task type, removing code documentation (JavaDocs), obfuscating obvious function and variable names, and adopting conventions and writing styles from the most recent Java version (at the time Java 18).

The final selection of code snippets is presented in Table 4. For the actual snippets, along with any modifications made, and the randomized order assigned to participants refer to our replication package.

Table 4 Final snippet selection with description, lines of code, and total number of inline code comments.

Snippet	Description	LOC	(#Comments)
1	Identifies a peak element in an array.	18	(+7)
2	Finds two elements that sum to a target.	19	(+8)
3	Computes maximum zero-sum subarray length.	19	(+10)
4	Longest consecutive sequence in array.	24	(+9)
5	Longest common subsequence between strings.	21	(+10)
6	Longest increasing subsequence length.	22	(+10)
7	Efficient power calculation.	17	(+7)
8	Fibonacci number at given position.	18	(+8)
9	Lists primes up to a number (Sieve of Eratosthenes).	18	(+6)
10	Binary search in sorted array.	21	(+7)
11	Counts palindromic substrings.	19	(+10)
12	Anagram check for two strings.	16	(+5)

Comment Design and Quality All comments were written by the first author and independently reviewed by two team members to ensure their quality, relevance, and clarity within the code context. This establishes a consistent baseline quality across all comments.

We employed both left-target and parent-target inline comments, as defined in Table 2, specifically because their consistent placement on the right side of the code ensured a clear spatial separation between code and comments. This separation facilitated a clearer distinction in the eye-tracking data, enabling more precise analysis of visual attention. While block comments are also prevalent in practice, they often refer to larger code sections and lack a one-to-one mapping with individual lines or expressions. This would have complicated both our gaze mapping and interpretation of fine-grained reading behavior. Investigating the effects of block comments remains an important direction for future research.

We deliberately included comments for nearly every line of code to provide consistent and dense coverage. While this dense coverage may depart from practices, where comments are more selectively applied, it was necessary to explore the effects of comment placement on comprehension and gaze behavior.

4.6 Post-Questionnaire

Aligned with the research questions of our study, we designed the post-questionnaire to gather in-depth insights into the participants' experiences and perspectives on the influence of comments on program comprehension. This alignment ensured that the questions directly correlated with the core themes of our study, namely understanding the impact of comments on program comprehension performance, visual attention, linearity of reading order, gaze strategies, and overall comment contribution in programming tasks.

The post-questionnaire consisted of two parts. In the first part, participants rated the code snippets they had encountered during their trial run using Likert scales. Specifically, they evaluated each snippet's perceived difficulty and the helpfulness of the accompanying comments. In the second part of the questionnaire, participants provided written responses (i.e., free-text answers) to open-ended questions (see Table 5), allowing them to articulate their thoughts in their own words. To provide a structured and consistent framework for these responses, we divided the questions into sections, each targeting specific aspects of the study's focus. This organization facilitated the correlation of subjective responses with the quantitative data gathered during the experiment's first phase, thereby enriching the thematic analysis with nuanced qualitative insights.

Table 5 Post-questionnaire questions and their relationship to our research questions.

Research Question	**Questions**
RQ_1: Program Comprehension Performance	How did the presence of comments impact your understanding of the code? Please describe your experience with the code snippets that contained comments.
$RQ_{2.1}$: Visual Attention	Were there any instances where your visual attention was drawn to comments within the code snippets? How did the presence of comments influence your gaze patterns while comprehending the code?
$RQ_{2.2}$: Linearity of Reading Order	Did the presence of comments affect the order in which you read and interpret the code elements? Please explain how comments may have influenced the flow of your reading.
$RQ_{2.3}$: Gaze Strategies	Did you adopt a specific gaze strategy while navigating through code snippets containing comments? How did the presence of comments influence your choices in directing visual attention between code and comments?
RQ_3: Role of Comments	From your perspective, how do comments contribute to your overall understanding of the code snippets? Do you find comments helpful in clarifying complex code segments or guiding your comprehension?
General Impressions	Please share any additional observations, insights, or thoughts you have regarding the role of comments in program comprehension.

4.7 Participants and Sampling Strategy

We recruited 20 computer science students from Saarland University, Germany, using convenience sampling. Participants were required to have a solid foundation in Java and core programming concepts such as recursion, algorithms, and data structures. Eligibility was assessed via a pre-study questionnaire collecting demographic information and programming experience to ensure a minimum level of proficiency.

The group comprised 18 males and 2 females, reflecting a gender distribution that aligns with typical patterns in the field of computer science at our university. The mean age of the participants was 25.8 ± 4.20 years. Educational backgrounds ranged from high school diploma (11 participants) to Bachelor's (7) and Master's degrees (2). The participants indicated varying Java expertise: 12 had practical project experience, 5 had basic knowledge, 2 used Java regularly, and 1 identified as an expert.

Figure 3 depicts the age distribution, Java programming experience, and overall programming experience in years among the participants.

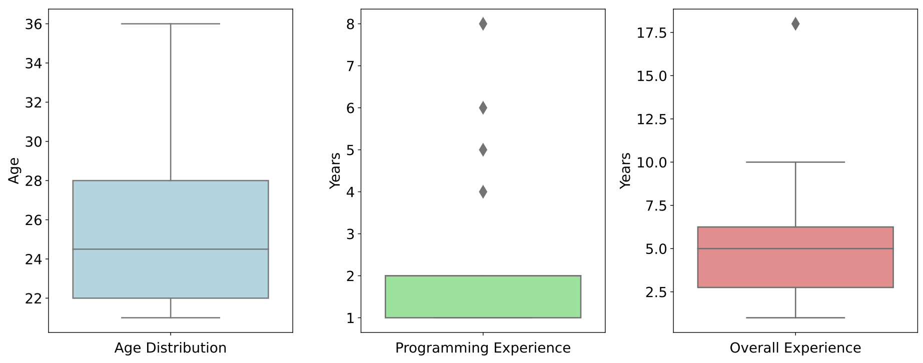


Fig. 3 Participants' age, Java programming experience, and overall programming experience in years.

4.8 Eye-Tracking and Data Collection

In our study, we used the Tobii EyeX[4] eye tracker to collect gaze data at a frequency of 60 Hz. Prior to the study, participants completed a calibration procedure, during which they looked at a series of points displayed on the screen. This calibration process allowed the eye tracker to accurately track their gaze data during the study.

We used a custom C# program to operate the eye tracker and guide participants through the study, including the calibration process and the presentation of code snippets. Throughout the study, the program collected responses from participants, completion times, and gaze data.

[4] https://help.tobii.com/hc/en-us/articles/212818309-Specifications-for-EyeX

4.9 Ethical Considerations

We obtained informed consent from all participants prior to their participation. We further provided them with detailed information about the nature of our study, the procedures involved, and their right to withdraw from the study at any time without any negative consequences. We anonymized the collected data.

5 Data Analysis Methodology

This section outlines the methodology of managing and refining the data. We describe the process for both behavioral and eye-tracking data, from the collection of raw data to the preparation for analysis.

5.1 Behavioral Data Analysis

For the behavioral data analysis, each response was manually evaluated to determine its semantic correctness. We considered responses with minor formatting inaccuracies, such as variations in decimal places, still semantically correct.

To balance correctness and completion time in a single interpretable metric, we computed a composite score per trial that equally weighs the accuracy of the participants' responses and the normalized completion time:

$$\text{Score} = 0.5 \times \text{correctness} + 0.5 \times \left(1 - \frac{\text{time} - \text{min\_time}}{\text{max\_time} - \text{min\_time}}\right) \times 100$$

In this formula, 'correctness' represents the correctness of the snippet responses (0 for incorrect, 1 for correct), and 'time' is normalized against the common minimum and maximum completion times for each snippet, considering both CM and CP conditions. This normalization ensures that the time scores for CM and CP conditions of the same snippet are comparable. A lower time value results in a higher score. This approach allowed us to create a balanced metric that encapsulates both the accuracy of responses and the efficiency of snippet completion under each condition.

5.2 Eye-Tracking Data Analysis

The eye-tracking data underwent several preprocessing steps to ensure data quality and reliability. The data analysis pipeline employed in our study is based on custom-developed code from Peitek et al. [73], which we extended and adapted to suit the specific requirements of this study. The pipeline encompasses various stages, including preprocessing, analysis of general metrics, analysis of AOI metrics, statistical analysis, and data visualization. The subsequent subsections detail each stage of the pipeline, providing a comprehensive overview of the data analysis process and its significance for our study.

5.2.1 Data Processing

The first stage of the pipeline is preprocessing the raw data, which is essential for ensuring data quality, reducing noise, and preparing the data for subsequent analyses. First, we performed essential data cleaning operations, including removing unnecessary columns and dropping rows related to rest conditions. We also adjusted and rounded timestamps, gaze positions, and experiment time values to facilitate further analysis. Next, we performed a series of operations to enhance data quality and extract additional features, including dropping duplicate timestamps, scaling the gaze data based on screen height, smoothing the gaze positions using a Savitzky-Golay filter (window length of 5, polynomial order of 3), and calculating the velocity of eye movements. Finally, we classified gaze events (fixations and saccades) using a velocity-based algorithm with a velocity threshold of 150 pixels per 100 milliseconds (cf. Section 2.3.1). If the velocity was below the threshold, it was interpreted as a fixation; if it exceeded the threshold, it was interpreted as a saccade. We computed relevant statistics for each fixation and saccade, including average position, time duration, frame count, average velocity, and distance traveled.

5.2.2 Analysis of General Metrics

Following the preprocessing stage, the pipeline proceeds to analyze general metrics obtained from the eye-tracking data:

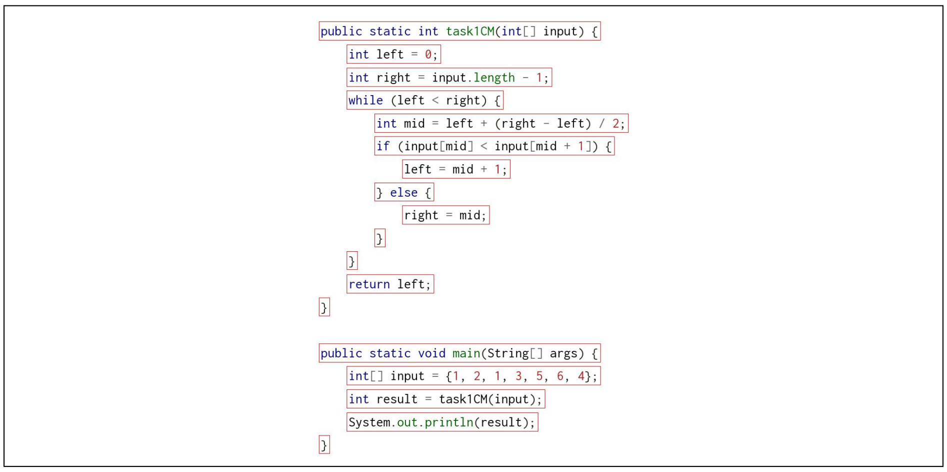
Step I Fixation and Saccade Analysis: Various statistics related to fixations and saccades are computed, including the number of fixations and saccades, fixation and saccade rates per second, and average fixation length and saccade distance. These statistics provide relevant insights into the temporal aspects and spatial characteristics of participants' eye movements.
Step II Snippet-Level Metrics: The analysis pipeline also considers snippet-level metrics by examining the eye-gaze behavior within specific snippets. Fixation and saccade data for each snippet are collected, and metrics such as fixations per second, saccades per second, fixation length, and saccade distance are computed. This analysis provides a more detailed understanding of participants' eye movements during different snippets.

5.2.3 Analysis of AOI Metrics

In addition to general metrics, the pipeline also includes the analysis of AOI metrics. By evaluating participants' gaze behavior within these AOIs, insights into the specific areas or elements that attract visual attention can be gained. The analysis of AOI metrics involved computing various metrics within the eye-gaze data. The following details the steps involved in this process:

Step I Fixation and Saccade Data Extraction: The analysis begins by extracting fixation and saccade data for each participant and snippet. Fixations and saccades associated with the specified snippet are retrieved and stored for further analysis.
Step II Creating AOIs: For addressing $RQ_{2.2}$, each line within the snippet is considered as an individual AOI. This approach enables the computation of metrics

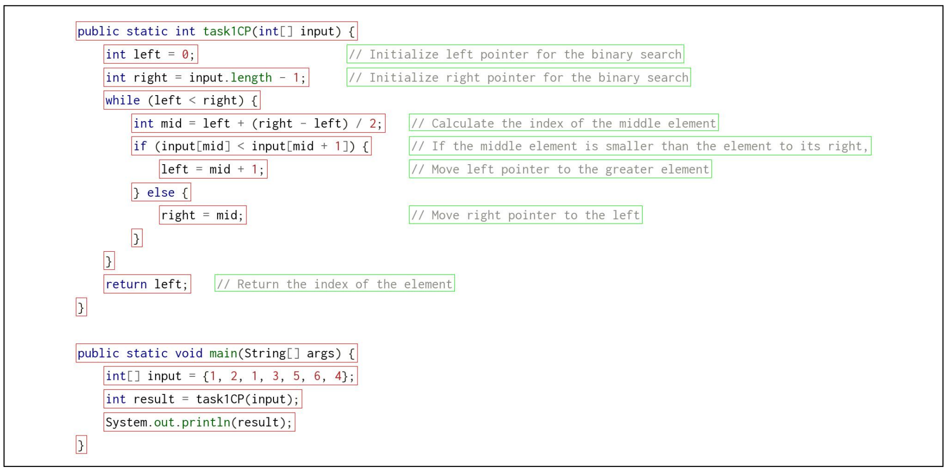
that characterize the linearity of reading order, following the methodology proposed by Busjahn et al. [21]. Additionally, by distinguishing between code and comment AOIs, we aim to examine the reading flow between these elements and identify the employed gaze strategies. For a better visualization of the AOIs, Figure 4 shows an example snippet with AOI overlays, where each line is represented as an individual AOI. Additionally, Figure 5 illustrates AOI overlays with a distinction between code and comment elements.



```java
public static int task1CM(int[] input) {
    int left = 0;
    int right = input.length - 1;
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (input[mid] < input[mid + 1]) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }
    return left;
}

public static void main(String[] args) {
    int[] input = {1, 2, 1, 3, 5, 6, 4};
    int result = task1CM(input);
    System.out.println(result);
}
```

Fig. 4 AOI Overlays with each code line as AOI for analyzing the linearity of reading order.



```java
public static int task1CP(int[] input) {
    int left = 0;                          // Initialize left pointer for the binary search
    int right = input.length - 1;          // Initialize right pointer for the binary search
    while (left < right) {
        int mid = left + (right - left) / 2;   // Calculate the index of the middle element
        if (input[mid] < input[mid + 1]) {     // If the middle element is smaller than the element to its right,
            left = mid + 1;                    // Move left pointer to the greater element
        } else {
            right = mid;                       // Move right pointer to the left
        }
    }
    return left;    // Return the index of the element
}

public static void main(String[] args) {
    int[] input = {1, 2, 1, 3, 5, 6, 4};
    int result = task1CP(input);
    System.out.println(result);
}
```

Fig. 5 AOI Overlays with distinction between code and comment AOIs.

Step III Matching Fixations to AOIs: The next step involves matching the extracted fixations to the appropriate AOIs. Fixations falling within the boundaries

of an AOI are considered hits, and relevant information such as the AOI name, line number, and fixation duration are recorded. For all subsequent analyses based on AOIs, we filtered out all fixations outside of defined AOIs. Following Busjahn et al. [21], we included fixations with a maximum of a 100 pixel horizontal deviation (ca. 7–8 characters), as small AOIs can otherwise be easily missed (e. g., a closing bracket) and may distort the results.
Step IV Computing AOI Metrics: Once the fixations are matched to the AOIs, various AOI metrics are calculated based on the matched fixations. These metrics provide insights into participants' eye-gaze behavior within specific AOIs.

(a) **Visual Attention Metrics:** Through analyzing and filtering the fixation hits for the different AOIs—Code and Comments—we aggregated detailed data on where and how long participants focused their gaze while comprehending code snippets. This approach allows us to gain relevant insights into the cognitive load involved in code comprehension, particularly in terms of how comments either guide or distract visual attention. The data are segmented into various categories: All Fixations, AOI Fixations, Code Fixations, and Comment Fixations, each quantified in terms of count and duration. We present a breakdown of these metrics in Table 6.

Table 6 Description of visual attention metrics.

Metric	**Description**
All Fixations	Total number of fixations within the entire snippet.
AOI Fixations	Total number of fixations within AOIs.
Code Fixations	Total number of fixations on code AOIs.
Comment Fixations	Total number of fixations on comment AOIs.

(b) **Linearity Metrics:** This builds directly on the insights gained from the work of Busjahn et al. [21] and Peitek et al. [73]. The papers highlight the distinction between natural language text reading and source code reading, emphasizing that, while natural language is typically read linearly, source code reading often deviates from this linear pattern. To measure this, both local and global gaze-based measures are used to assess the degree of linearity in reading behavior. Our study replicates and extends this approach by employing five fixation-based local metrics: Vertical Next Fixations, Vertical Later Fixations, Regression Fixations, Horizontal Later Fixations, and Line Regression Fixations. These metrics reflect the immediate gaze patterns of participants during reading snippets. Additionally, we incorporate the global metrics established by the referenced studies. It utilizes the N-W algorithm for analyzing reading patterns in programming. This algorithm is employed in two forms: a naive and a dynamic calculation for both story and execution order of the code. The naive calculation compares the participant's reading order directly with the expected linear sequence of the code, while the dynamic calculation allows the sequence to be repeated multiple times accounting for the iterative nature of reading complex code. Details of these metrics are catalogued in Table 7.
In our setup, story and execution order vary between the snippet types, CP and CM. Table 8 visually represents the different AOI reading orders employed

Table 7 Description of linearity metrics.

	Metric	**Description**
Local	Vertical Next Text	% of forward saccades that either stay on the same line or move one line down.
	Vertical Later Text	% of forward saccades that either stay on the same line or move down any number of lines.
	Horizontal Later Text	% of forward saccades within a line.
	Regression Rate	% of backward saccades of any length.
	Line Regression Rate	% of backward saccades within a line.
	SaccadeLength	Average Euclidean distance between every successive pair of fixations.
Global	Story Order	N-W alignment score of fixation order with linear text reading order.
	Execution Order	N-W alignment score of fixation order with the program's control flow order.

for Snippet 1 using radial transition graphs. For the complete set of radial transition graphs refer to our replication package.

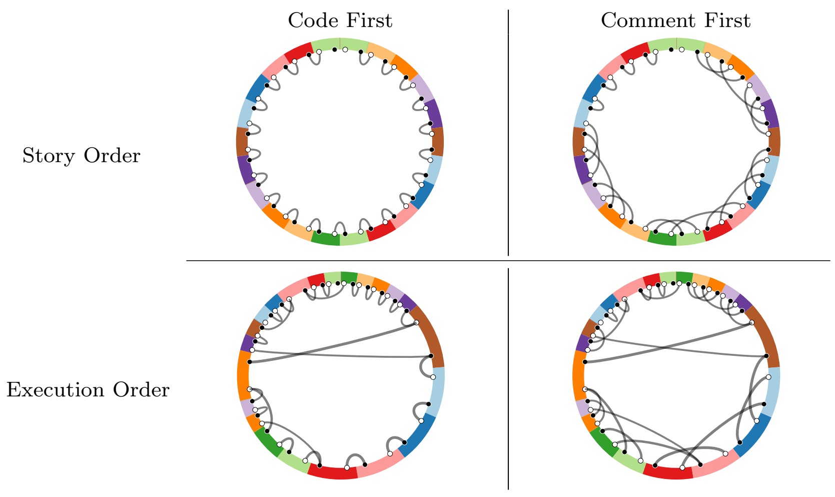


Table 8 Comparison of Story Order and Execution Order with Code First and Comment First strategies for Snippet 1 CP.

(c) **Gaze Strategy Metrics:** Our study extends the aforementioned linearity metrics by adopting a similar approach for investigating gaze strategies within CP snippets, where we aim to examine how participants shift their focus between code and comments during comprehension tasks. Local metrics, such as code-to-comment and comment-to-code ratios, provide insights into immediate gaze patterns and reveal how participants navigate between code and accompanying comments. For a broader perspective, we applied the N-W algorithm once more. This time, it was used to assess the alignment of participants' gaze patterns with pre-defined AOI sequences for Code-First and Comment-First reading orders. This global measure will help us understand the overarching gaze strategies adopted by participants in the context of different reading approaches. Details of these metrics are catalogued in Table 9.

Table 9 Description of gaze strategy metrics.

	Metric	Description
Local	CodeToComment	% of saccades that move from code to comment.
	CommentToCode	% of saccades that move from comment to code.
Global	Code-First	N-W alignment score of fixations with code-first reading order.
	Comment-First	N-W alignment score of fixations with comment-first reading order.

5.3 Statistical Analysis

Our study employs a partially crossover within-subject design, exposing participants to both Comments Missing (CM) and Comments Present (CP) conditions across varied code snippets. This imposes challenges related to data imbalance and the necessity to manage within-subject variability. In our analysis, we take a holistic statistical approach to assess the impact of CM and CP conditions on code snippet performance and behavior, capturing both overall trends and individual variations.

5.3.1 Wilcoxon Signed-Rank Test

In a first step, we employed the Wilcoxon Signed-Rank Test [123] for the analysis of aggregated data, focusing on the median differences between CM and CP conditions without the need to account for individual participant variations. This non-parametric test was appropriate for datasets not conforming to the normal distribution assumptions required by parametric tests. It allowed for a direct comparison of paired observations, such as the performance or behavior under CM and CP conditions, to discern statistically significant median differences across these conditions. This method was instrumental in identifying general trends and overarching effects of comments on code comprehension across the entire participant pool. The analysis typically adopted a two-sided approach to assess median differences between snippet conditions, ensuring an unbiased investigation. However, when a specific effect direction was hypothesized, we applied a one-sided test, tailoring the analysis to align with anticipated outcomes and theoretical predictions.

5.3.2 Linear Mixed-Effects Models (LMEs)

Following the broader insights gained from the Wilcoxon Signed-Rank Test, we fitted separate LMEs in cases where a more in-depth analysis was needed on a snippet-level, allowing for a more detailed analysis of various performance and eye-tracking metrics across the snippets. In these models, the snippet type was treated as a fixed effect, directly evaluating its influence on the measured outcomes. Importantly, individual differences among participants were modeled as random effects. This approach acknowledged and controlled for the individual differences in baseline performance levels across snippets. The models were defined as:

$$\text{Prediction}_{ij} = \beta_0 + \beta_1 \times \text{CP}_{ij} + u_j - \epsilon_{ij}$$

where Prediction_{ij} represents the metric entry for the i-th observation from the j-th participant, β_0 represents the intercept, β_1 is the coefficient assessing the effect of the CP condition (relative to the CM condition, which serves as the reference category), u_j denotes the random effect to capture individual participant differences, and ϵ_{ij} is the residual error. For instance, regarding correctness, the models predicted the binary outcome (correct or incorrect) based on the snippet type, and for time, they predicted the duration taken to complete each snippet. The significance of the CP condition's effect was then evaluated using p-values for the coefficient β_1, with lower values indicating a more significant impact of the condition on the respective response variable.

To further refine the analysis, we applied a false discovery rate (FDR) [80] correction whenever we conducted several statistical tests on the same dataset. We used the Benjamini-Hochberg [10] procedure, which adjusts p-values to reflect a more accurate significance level of the observed effects. This correction controls for the increased risk of Type-I errors associated with multiple comparisons, thereby ensuring the statistical integrity of our findings.

This combination of the Wilcoxon Signed-Rank Test and LMEs provided a comprehensive methodological framework for the study. By employing the Wilcoxon test, the study could ascertain broad, overarching trends, establishing a baseline understanding of the effects of snippet conditions. Subsequently, LMEs offered a deeper dive into these effects across the snippets while accommodating individual participant strategies.

5.4 Qualitative Analysis of Ratings

Following the quantitative phase of our study, we conducted a lightweight thematic analysis of the responses collected from the post-questionnaire (cf. Section 4.6) to discern the broader implications of our findings. This analysis was guided by Braun and Clarke's six-phase framework, providing a systematic approach to understanding the nuanced insights of how comments influence program comprehension from a programmer's perspective [58].

1. *Familiarization with the Data:* We engaged with the questionnaire data through repeated reading and note-taking to grasp the depth of responses.
2. *Generating Initial Codes:* We systematically coded the data to identify significant phrases, patterns, or concepts emerging directly from the responses.
3. *Searching for Themes:* We organized the codes into potential themes that encapsulate the core insights from the data.
4. *Reviewing Themes:* We refined these themes to ensure they accurately reflected the coded data and the entire dataset.
5. *Defining and Naming Themes:* We clarified the essence of each theme and how they interrelate, offering a coherent narrative of the data.
6. *Integration and Reporting:* We related the thematic findings back to the study's research questions and the broader context, illustrating how these qualitative insights enrich our quantitative analysis.

This structured analysis allowed for a comprehensive exploration of the qualitative data, enhancing our understanding of the implications of comments on program comprehension.

6 Results

In this section, we present our study's findings. First, we analyze the behavioral data to explore the impact of comments on program comprehension. Subsequently, our analysis shifts to eye-tracking data, focusing on aspects such as visual attention, the linearity of reading order, and gaze strategies. By combining eye tracking with behavioral data (correctness and time), we explore how comments affect the understanding and eye-movement patterns of computer science students during program comprehension. The structure of this section aligns with the five research questions outlined in Section 4.2:

RQ_1: Program comprehension performance: The first research question investigates program comprehension performance by analyzing correctness rates and completion times to understand the effect of comments on program comprehension.

RQ_2: Reading Behavior: The second research question focuses on reading behavior during program comprehension and is addressed through three sub-questions:

$RQ_{2.1}$: Visual attention: The first sub-question examines visual attention by assessing eye-tracking data, including fixation counts and durations, to gain insights into the allocation of visual attention in the presence and absence of comments.

$RQ_{2.2}$: Linearity of reading order: The second sub-question investigates the linearity of reading order by analyzing reading patterns and how they are influenced by the presence of comments, using eye-tracking measures and an adapted version of the Needleman–Wunsch (N-W) algorithm.

$RQ_{2.3}$: Gaze strategies: The third sub-question explores gaze strategies by examining how programmers navigate between code and comments, using both local and global metrics to characterize these strategies.

RQ_3: Participants' self-reported opinions: The third research question addresses the self-reported opinions of participants through an analysis of their subjective ratings on the difficulty of code snippets and the perceived contribution of comments to program comprehension.

This section focuses on presenting key results that are most relevant to the overarching objectives of our study. We provide a thorough discussion of these results, including their implications and potential interpretations in Section 7.

6.1 Effect of Comments on Program Comprehension Performance

In our study, we conducted a comprehensive evaluation of the effect of comments on program comprehension performance. Specifically, we analyzed the correctness and completion time of participants' responses to the 12 code snippets under two distinct conditions: Comments Missing (CM) and Comments Present (CP).

Our study reveals a complex and diverse impact of comments on the comprehension of different code snippets, as evidenced by varying correctness rates, completion times, and overall scores. To illustrate this point, we concentrate on four snippets, each representing a unique combination of outcomes:

– Snippet 1, which involved identifying a peak element in an array, showed a decrease in performance with comments present (CP). The correctness rate dropped from 70% to 30%, and the average completion time increased from 190.9 to 271.7 seconds. Correspondingly, the overall score decreased by 30% in the CP condition, indicating a negative impact of comments.
– Conversely, Snippet 9, listing primes up to a given number, led to a substantial improvement with comments. The correctness rate rose from 60% under CM to 100% under CP, and the average completion time decreased from 339.2 to 189.4 seconds. This positive effect is further reflected in the overall score, which increased by 34% under CP.
– Snippet 5, dealing with finding the longest common subsequence between two strings, showed an increase in correctness from 50% in CM to 70% in CP. However, this was offset by a longer completion time. Despite this, the overall score increased by 7% in the CP condition, suggesting a net positive effect of comments.
– Snippet 11, counting palindromic substrings, showed no variation in correctness (10%) between the two conditions. Nonetheless, there was a notable decrease in completion time under CP, and the overall score increased by 8%, highlighting the efficiency gains due to comments.

We present the detailed results in Table 10, contrasting the effects of Comments Missing (CM) and Comments Present (CP) snippets on correctness rates and completion times. Additionally, Figure 6 visually complements these findings by graphically illustrating the effects captured in the table. By examining the distance of each snippet from the neutral benchmark line in the plots, we can visually infer the overall effect of comments. This visual analysis corroborates the numerical scores from the table, providing an intuitive understanding of how comments influence correctness and time efficiency.

Table 10 Comparative analysis of correctness and completion time of CM and CP snippets. The overall effect was determined by equally weighting correctness and normalized time (cf. Section 5.1). Correctness and time cells are shaded in blue, scaled proportionally to the maximum value in each column. The overall effect column uses orange to indicate a negative effect and green for a positive effect.

Snippet	Description	Type	Correctness	Time (in sec.)	Overall Effect
1	Identifies a peak element in an array.	CM	70.0%	190.9 ± 87.6	-30.0%
		CP	30.0%	271.7 ± 74.1	
2	Finds two elements that sum to a target.	CM	60.0%	206.4 ± 100.4	-10.2%
		CP	50.0%	241.4 ± 81.3	
3	Maximum zero-sum subarray length.	CM	50.0%	238.0 ± 71.1	-18.7%
		CP	30.0%	291.0 ± 86.4	
4	Longest consecutive sequence in array.	CM	50.0%	203.0 ± 90.3	-11.4%
		CP	40.0%	244.3 ± 90.2	
5	Longest common subsequence between strings.	CM	50.0%	295.5 ± 156.8	7.0%
		CP	70.0%	331.3 ± 151.5	
6	Longest increasing subsequence length.	CM	40.0%	333.4 ± 240.1	-2.3%
		CP	30.0%	285.3 ± 93.0	
7	Efficient power calculation.	CM	40.0%	229.1 ± 85.2	-1.1%
		CP	40.0%	238.9 ± 121.9	
8	Fibonacci number at given position.	CM	60.0%	230.1 ± 71.8	11.3%
		CP	60.0%	167.3 ± 80.8	
9	Sieve of Eratosthenes.	CM	60.0%	339.2 ± 107.9	34.1%
		CP	100.0%	189.4 ± 97.1	
10	Binary search in sorted array.	CM	80.0%	169.5 ± 65.0	9.4%
		CP	90.0%	144.6 ± 92.4	
11	Counts palindromic substrings.	CM	10.0%	276.5 ± 101.6	8.2%
		CP	10.0%	198.8 ± 103.4	
12	Anagram check for two strings.	CM	90.0%	62.9 ± 20.8	2.4%
		CP	100.0%	69.7 ± 44.7	

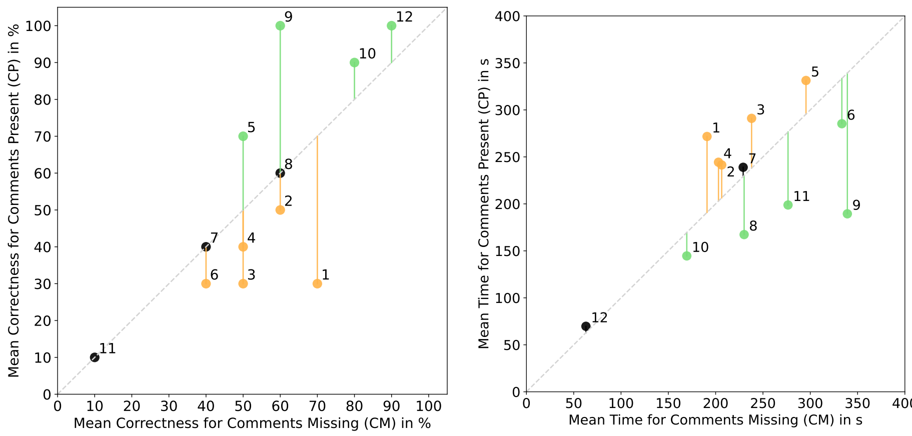


Fig. 6 Effect of comments on mean correctness and time across snippets. Green signifies positive impacts, orange denotes negative impacts, and black indicates a neutral effect.

When considering all snippets together, a nuanced picture emerges:

– *Positive impact of comments:* Snippets 5, 8, 9, 10, and 11 showed a beneficial effect of comments, as evidenced by their net positive scores and distances to the neutral benchmark in both plots. These snippets showed notable improvements in correctness and/or efficiency, suggesting that comments can significantly enhance understanding and performance in specific contexts.
– *Negative impact of comments:* Snippets 1, 2, 3, and 4 demonstrate a negative impact of comments, characterized by lower correctness and longer completion times, as seen through their negative overall scores and relative positions to the neutral line.
– *Neutral/mixed effect of comments:* Snippets 6, 7, and 12 present a more ambiguous effect, with slight variations in performance that do not strongly trend towards positive or negative outcomes. For Snippet 7, both correctness and time metrics indicate a neutral effect.

To account for both within-subject and between-subject variations, we employed a linear mixed-effects model for each snippet individually (cf. Section 5.3.2). This approach allowed us to consider the fixed effects of snippet type (CM and CP), while incorporating random effects to accommodate inter-participant variability.

The analysis revealed varying effects of comments on correctness and time across different snippets. In some snippets, the CP condition showed a significant difference in performance metrics compared to the CM condition, whereas, in others, the differences were not statistically significant. These results further indicate that the influence of snippet type on performance is snippet-dependent. We present the results from these models in Table 11.

Table 11 LME results for correctness and time of each snippet. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	Correctness			Time		
Snippet	Intercept	Effect of CP	*p*-Value	Intercept	Effect of CP	*p*-Value
1	0.7	-0.4	0.059	190.87	80.87	**0.002**
2	0.6	-0.1	0.544	206.43	34.98	0.173
3	0.5	-0.2	0.211	238.03	52.96	**0.008**
4	0.5	-0.1	0.544	203.01	41.26	0.148
5	0.5	0.2	0.211	295.52	35.80	**0.017**
6	0.4	-0.1	0.651	333.44	-48.15	0.439
7	0.4	0.0	1.000	229.06	9.79	0.662
8	0.6	0.0	1.000	230.08	-62.76	$<$**0.001**
9	0.6	0.4	$<$**0.001**	339.24	-149.85	$<$**0.001**
10	0.8	0.1	0.514	169.47	-24.88	0.204
11	0.1	0.0	1.000	276.46	-77.70	**0.002**
12	0.9	0.1	0.157	62.92	6.75	0.637

RQ_1 Our results reveal that the presence of comments can exert diverse effects on program comprehension performance. While some snippets benefit extremely from comments in terms of both time and correctness, others suffer from longer completion times without a corresponding rise in correctness. These variations highlight a complex and snippet-specific interplay between comments and performance, suggesting that the efficacy of comments is highly contextual, enhancing comprehension and efficiency in some scenarios while potentially hindering them in others.

6.2 Effect of Comments on Visual Attention

Building on the behavioral results of RQ_1, we now turn our focus to a more detailed investigation of how these behaviors manifest in the visual attention during program comprehension, a pivotal aspect of understanding how programmers interact with commented code.

Our study reveals a significant shift in visual attention when comments are present. In snippets where comments are absent, all AOI fixations are focused on the code. However, with the introduction of comments, about 23% of visual attention has been allocated to them. This was determined by calculating the proportion of fixations on comments relative to the total number of fixations across all areas of interest—specifically, 92 out of 397 fixations were directed at comments (cf. Table 12). Consequently, the focus on code drops to around 77%. This significant diversion of attention highlights that comments play a critical role in shaping programmers' visual engagement with code. In snippets where comments are present, participants are dedicating a substantial portion of their attention to them.

To illustrate these findings more concretely, Table 12 encapsulates the aggregated results of our study. It compares the metrics of visual attention (cf. *Step IV* a in Section 5.2.3) in snippets with Comments Missing (CM) against those with Comments Present (CP).

Furthermore, we quantitatively analyzed the fixation counts and durations for the different categories across the different snippets to capture possible snippet-specific effects. We again employed LMEs for each snippet to rigorously evaluate these differences. The models were constructed similar to the previous section to assess the fixed effects of the snippet type (CM or CP) on visual attention, while controlling for random effects due to individual participant variations (cf. Section 5.3). This approach allowed us to draw statistically robust conclusions about the role of comments in directing visual attention during program comprehension on a snippet-specific level.

We present the results of these analyses in Table 13 (All Fixations), Table 14 (AOI Fixations), Table 15 (Code Fixations), and Table 16 (Comment Fixations). Table 13 demonstrates notable findings. For Snippets 1 to 4, there are significant increases in both fixation counts and durations when comments are present. This result aligns with expectations, considering that the inclusion of comments adds to the total text visible on the screen. However, the pattern observed in these snippets is not universally consistent. For the subsequent snippets, the impact of

Table 12 Comparative analysis of visual attention metrics of CM and CP snippets. Fixation counts and durations represent mean values aggregated across all participants and snippets per condition. Bar widths are proportional to the underlying values.

		Type		Wilcoxon Signed-Rank Test			
		CM	CP	Effect Direction	Statistic	*p* -Value	Corr. *p* -Value
All Fixations	Count	525	579	two-sided	2867.0	**0.046**	0.067
	Duration	177.0	171.4	two-sided	3620.0	0.979	0.979
AOI Fixations	Count	358	397	two-sided	2881.5	**0.050**	0.067
	Duration	123.9	121.2	two-sided	3582.5	0.901	0.979
Code Fixations	Count	358	305	less	2732.5	**0.018**	**0.037**
	Duration	123.9	100.3	less	2468.0	**0.001**	**0.003**
Comment Fixations	Count	-	92	greater	7260.0	$<$ **0.001**	$<$ **0.001**
	Duration	-	20.9	greater	7260.0	$<$ **0.001**	$<$ **0.001**

comments varies. In some cases, there is no significant difference in fixation counts between the two conditions, while in others, particularly Snippets 8, 9, and 10, an interesting trend emerges. In these snippets, not only does the fixation count decrease with the presence of comments, but there is also a notable reduction in the duration of fixations per participant. This suggests that comments, in certain contexts, may actually streamline the process of visual engagement, possibly making the code easier to comprehend or navigate.

Table 13 LME results for all fixation counts and durations of each snippet. Lower numbers are generally preferable and suggest less cognitive load. The cell shading highlights a significant effect of comments (**positive** or **negative**). Values marked with an asterisk (*) indicate the use of alternate fitting methods ('powell', 'lbfgs') due to convergence issues, where the determinant of matrices approached zero, leading to linear algebra errors.

	All Fixations							
	Count				Duration			
Snippet	Intercept	Effect of CP	*p*-Value	Corr. *p*-Value	Intercept	Effect of CP	*p*-Value	Corr. *p*-Value
1	469.8	191.7	0.003	**0.004**	149.362	56.270	<0.001	**<0.001**
2	480.4	217.9	0.006	**0.013**	154.885	28.577	0.108	0.123
3	557.2	285.9	0.001	**0.002**	177.034	44.258	<0.001	**<0.001**
4	497.0	239.1	0.002	**0.003**	160.651	30.799	<0.001	**<0.001**
5	722.6	64.8	<0.001	**<0.001**	219.540	36.580	0.292	0.467
6	642.1	68.6	0.617	0.729	245.289	-18.819	0.653	0.729
7	456.3	25.7	0.650	1.000	183.148	-0.723	0.930	1.000
8	441.4	-30.2	0.768	0.768	163.040	-39.757	<0.001	**<0.001**
9	833.4	-351.2	<0.001	**<0.001**	269.014	-125.487	<0.001	**<0.001**
10	418.4	-35.8	0.689	0.689	138.568	-25.288	0.031	**0.049**
11	*610.0	*-56.0	*0.685	*0.685	214.397	-59.307	0.122	0.195
12	172.7	27.4	0.504	0.806	49.439	5.230	0.450	0.806

Table 14 presents data on fixations specifically targeting the designated AOIs for code and comments. We observe a similar trend of variability as before, although not as pronounced. The data indicate that the presence of comments influences where participants focus their attention, but the effect varies across snippets. The impact of comments becomes more distinct when analyzing Tables 15 and 16,

Table 14 LME results for AOI fixation counts and durations of each snippet. Lower numbers are generally preferable and suggest less cognitive load. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	AOI Fixations							
	Count				Duration			
Snippet	Intercept	Effect of CP	p-Value	Corr. p-Value	Intercept	Effect of CP	p-Value	Corr. p-Value
1	314.7	133.0	<0.001	**<0.001**	97.9	40.9	0.066	0.075
2	333.4	178.8	<0.001	**<0.001**	108.4	25.3	0.060	0.081
3	324.8	228.4	<0.001	**<0.001**	102.7	43.1	<0.001	**<0.001**
4	321.8	127.8	0.054	0.087	104.5	18.9	0.417	0.477
5	534.2	75.6	0.477	0.635	174.8	35.5	0.223	0.447
6	377.4	78.2	0.280	0.559	141.8	9.9	<0.001	**<0.001**
7	334.2	8.2	0.876	1.000	145.5	-7.0	0.580	1.000
8	339.0	-41.8	0.124	0.142	133.5	-40.1	0.007	**0.011**
9	564.7	-250.2	<0.001	**<0.001**	191.9	-92.5	<0.001	**<0.001**
10	298.5	-72.7	0.229	0.262	100.9	-31.4	0.064	0.086
11	445.5	-10.8	0.640	0.853	156.4	-34.8	0.289	0.463
12	112.1	4.9	0.830	0.949	28.7	-0.1	0.994	0.994

Table 15 LME results for code fixation counts and durations of each snippet. Lower numbers are generally preferable and suggest less cognitive load. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	Code Fixations							
	Count				Duration			
Snippet	Intercept	Effect of CP	p-Value	Corr. p-Value	Intercept	Effect of CP	p-Value	Corr. p-Value
1	314.7	74.9	<0.001	**<0.001**	97.9	30.4	0.270	0.270
2	333.4	105.4	0.045	0.072	108.4	10.2	0.499	0.499
3	324.8	20.4	<0.001	**<0.001**	102.7	-9.5	0.509	0.582
4	321.8	59.6	0.324	0.432	104.5	5.4	0.814	0.814
5	534.2	-63.0	0.557	0.637	174.8	4.4	0.890	0.890
6	377.4	-24.5	0.729	0.729	141.8	-12.4	0.535	0.729
7	334.2	-68.3	0.142	0.567	145.5	-27.2	0.219	0.583
8	339.0	-118.2	0.029	**0.038**	133.5	-54.5	<0.001	**<0.001**
9	564.7	-326.1	<0.001	**<0.001**	191.9	-109.7	<0.001	**<0.001**
10	298.5	-105.9	<0.001	**<0.001**	100.9	-37.4	0.008	**0.017**
11	445.5	-189.8	<0.001	**<0.001**	156.4	-80.0	0.001	**0.002**
12	112.1	-10.7	0.698	0.931	28.7	-2.5	<0.001	**<0.001**

which categorize fixations based on their focus on code versus comments. An interesting observation from these tables is the shift in visual attention from code to comments for Snippets 5 to 11. In these snippets, we find that the inclusion of comments corresponds to a decrease in both the count and duration of fixations on code elements, with some snippets showing significant reductions. Simultaneously, there is an increase in fixation counts and durations on the comments. This pattern suggests that comments, in these instances, may be redirecting attention away from the code to themselves, possibly providing clarifications or additional information that makes understanding the code easier. This effect underscores the potential of comments in guiding visual attention and influencing the comprehension process in programming tasks.

Table 16 LME results for comment fixation counts and durations of each snippet. Lower numbers are generally preferable and suggest less cognitive load. The cell shading highlights a significant effect of comments (**positive** or **negative**). Values marked with an asterisk (*) indicate the use of alternate fitting methods ('powell', 'lbfgs') due to convergence issues, where the determinant of matrices approached zero, leading to linear algebra errors.

	Comment Fixations							
	Count				Duration			
Snippet	Intercept	Effect of CP	p-Value	Corr. p-Value	Intercept	Effect of CP	p-Value	Corr. p-Value
1	0.0	58.1	<0.001	**<0.001**	0.0	10.5	<0.001	**<0.001**
2	0.0	73.4	<0.001	**<0.001**	0.0	15.1	<0.001	**<0.001**
3	*0.0	*208.0	*<0.001	***<0.001**	0.0	52.6	<0.001	**<0.001**
4	0.0	68.2	<0.001	**<0.001**	0.0	13.5	<0.001	**<0.001**
5	0.0	138.6	<0.001	**<0.001**	0.0	31.0	<0.001	**<0.001**
6	0.0	102.7	<0.001	**<0.001**	0.0	22.3	<0.001	**<0.001**
7	*0.0	*76.5	*<0.001	***<0.001**	0.0	20.2	<0.001	**<0.001**
8	0.0	76.4	<0.001	**<0.001**	0.0	14.4	<0.001	**<0.001**
9	0.0	75.9	<0.001	**<0.001**	0.0	17.2	<0.001	**<0.001**
10	0.0	33.2	<0.001	**<0.001**	0.0	6.0	<0.001	**<0.001**
11	*0.0	*179.0	*<0.001	***<0.001**	0.0	45.2	<0.001	**<0.001**
12	0.0	15.6	<0.001	**<0.001**	0.0	2.4	<0.001	**<0.001**

RQ$_{2.1}$ Our study's findings reveal a significant impact of comments on visual attention during program comprehension. We discovered that the presence of comments redirects approximately 23% of visual attention away from the code, suggesting a substantial shift in focus towards the comments. This reallocation of attention from code to comments underlines the influential role comments play in the cognitive process of understanding code (for the better or the worse). These insights, derived from detailed analysis of fixation counts and durations across Code and Comment AOIs, reveal a nuanced dynamic in how programmers interact with and process information in commented code.

6.3 Effect of Comments on Linearity of Reading Order

Having established the significant role of visual attention in program comprehension, as evidenced by the substantial focus programmers dedicate to comments, we now extend our exploration to another critical dimension: the linearity of reading order in programming tasks, as described in *Step IV*b of Section 5.2.3.

6.3.1 Local Metrics

Examining the impact of comments on the local linearity of reading order in program comprehension, our analysis unveils nuanced effects. The presence of comments significantly reduces the Vertical Later Fixations, suggesting a more linear, top-to-bottom reading pattern with comments, with participants less likely to skip over multiple lines. Conversely, we observe an increase in Horizontal Later Fixations, suggesting that comments encourage more lateral scanning within a line. Interestingly, the Regression Rate diminishes in the presence of comments, pointing to a reduction in backward saccades, thereby streamlining the reading process. This trend is combined with an increased Line Regression Rate, hinting at more frequent revisits within a line, potentially to reconcile code understanding with accompanying comments. Notably, the Saccade Length remains consistent across both scenarios, underscoring a uniformity in the distance of eye movements regardless of comment inclusion.

Table 17 presents our findings, offering a statistical analysis of various metrics such as Vertical Next and Later Fixations, Regression Fixations, Horizontal Later Fixations, Line Regression Fixations, and Saccade Length.

Table 17 Comparative analysis of local linearity metrics of CM and CP snippets.

	Type		Wilcoxon Signed-Rank Test		
Metric	CM	CP	Statistic	p -Value	Corr. p -Value
Vertical Next Fixations	14.4%	14.3%	14.0	1.000	1.000
Vertical Later Fixations	14.3%	12.1%	1.0	**0.011**	**0.018**
Regression Fixations	28.7%	25.7%	2.0	**0.001**	**0.004**
Horizontal Later Fixations	16.6%	22.8%	0.0	$<$ **0.001**	**0.003**
Line Regression Fixations	14.8%	16.1%	3.0	**0.012**	**0.018**
Saccade Length (in pixels)	43.1	41.6	29.0	0.470	0.564

6.3.2 Global Metrics

Next, we present the results on the impact of comments on global metrics of code reading. To assess the alignment between participants' reading behavior and the different reading orders, we adopt global reading metrics based on prior work (cf. *Step IV*b of Section 5.2.3). These metrics are derived using the N-W sequence alignment algorithm, which quantifies how closely a participant's observed reading order follows an expected reference order. We compute two variants of this metric: a naïve and a dynamic version.

– The naïve global metric assumes a singular reading process. It directly compares the participant's observed line reading order to the reference sequence (either the story order or the execution order) without permitting repetitions. This metric reflects how well a participant reads the code in a single, idealized pass.
– The dynamic global metric accounts for the inherently iterative nature of code comprehension. It allows lines to be revisited multiple times, comparing the participant's full sequence—including re-readings and jumps—to a potentially repeated version of the reference order. This more flexible metric captures realistic and complex reading patterns.

The N-W scores produced by these comparisons provide a quantitative measure of alignment: higher scores indicate closer conformity to the reference order, while lower scores suggest deviations. The `RTGCT` offers a complementary visual overview of how these reading patterns manifest. For example, Figure 7 shows the varied line reading orders observed for Snippet 1. Radial transition graphs summarizing the global reading orders for all code snippets are included in the replication package.

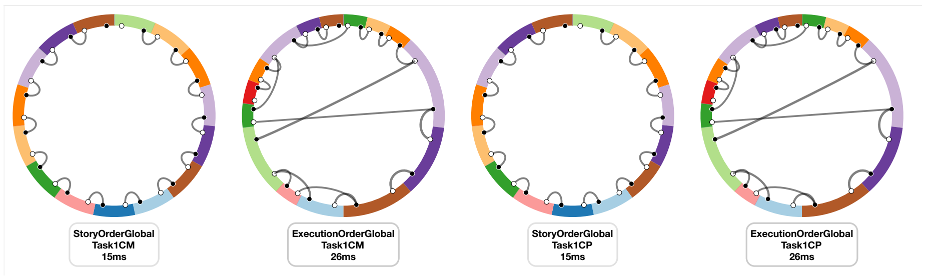


Fig. 7 `RTGCT` visualization of different line reading orders for Snippet 1.

Our aggregated findings show that the dynamic N-W scores (Story and Execution Order) consistently surpass the naive scores across snippets. This indicates that the N-W algorithm's dynamic adaptation better reflects participants' reading order, which usually includes several iterations of reading the same lines. Furthermore, the reading order mostly resembled the execution order of the code rather than the story order. Comments seem to have little to no impact on the N-W scores of the different metrics.

This is visually summarized in Figure 8, which presents the average global alignment scores across all snippets for both CM and CP versions. Each dashed line connects the score of a CM and CP pair for a particular metric. If the line is nearly vertical, it indicates minimal impact of comments; a shift to the right or left would imply improvement or deterioration, respectively. The lines across all reading orders support the conclusion that comments have a negligible global effect on how linearly participants read the code. This pattern is quantitatively corroborated by Table 18, which shows no significant differences in aggregated scores between CM and CP conditions.

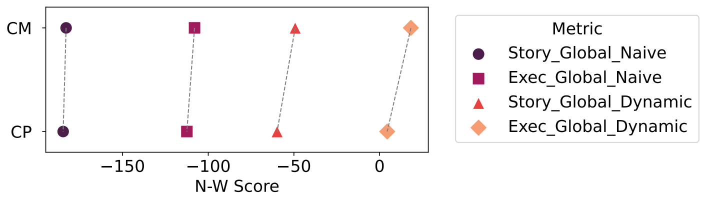


Fig. 8 Aggregated linearity N-W scores for CM and CP snippets. The figure illustrates the aggregate scores for both story and execution reading sequences using naive and dynamic N-W calculations.

A closer examination of the individual snippets presents a more nuanced picture. Figure 9 provides a visualization of global N-W alignment scores at the snippet level, comparing CM and CP versions across the four reading orders. Each dashed line again connects the CM and CP values for a specific snippet and metric. Negative slopes indicate an increase in alignment when comments are present, and positive slopes indicate a decrease in alignment with a particular order. The visualization highlights that the influence of comments is not uniform across all

Table 18 Comparative analysis of global linearity N-W scores of CM and CP snippets.

	N-W Score		Wilcoxon Signed-Rank Test		
Metric	CM	CP	Statistic	*p*-Value	Corr. *p*-Value
Story Order Naive	-182.95	-184.67	3451.0	0.639	0.639
Exec Order Naive	-107.99	-112.46	3226.5	0.362	0.483
Story Order Dynamic	-49.25	-59.80	2935.0	0.069	0.137
Exec Order Dynamic	18.27	4.46	2751.5	0.057	0.137

snippets but instead shows a contrasting pattern between two groups. For Snippets 1 to 6, the presence of comments mostly leads to a weaker alignment with the story order and mostly a stronger alignment with the execution order. In contrast, Snippets 7 to 12 show the opposite tendency. For this group of snippets, the comments show an increased alignment to the story order and decreased alignment to the execution order.

While comments in the first group of snippets appear to support a more execution-oriented understanding of the code, those in the second group seem to encourage a more linear, story-order interpretation. This observation indicates that comments can shift participants' reading strategies depending on the context and structure of the snippet. Interpreting this in the light of the results from RQ.1 (cf. Table 10), for snippets where comments had a more positive impact (Snippets 7 to 12), the comments enhanced the story order metric but adversely influenced execution order alignment. Essentially, when comments were actually beneficial, they made the code reading order more linear, whereas in Snippets 1 to 6, where comments had no or negative effect, the alignment with the story order mostly decreased.

This pattern is further supported by quantitative analysis. For Snippets 1 to 6, comments exhibit a predominantly negative effect on both naive story and execution order scores, with several of these effects reaching statistical significance. In contrast, Snippets 7 to 12 show a trend in the opposite direction, where comments are associated with more linear reading behavior, particularly in the dynamic story order. These findings are reinforced in Tables 19 and 20, which report the statistical outcomes and emphasize that the influence of comments on linearity is snippet-specific, rather than uniform across the dataset.

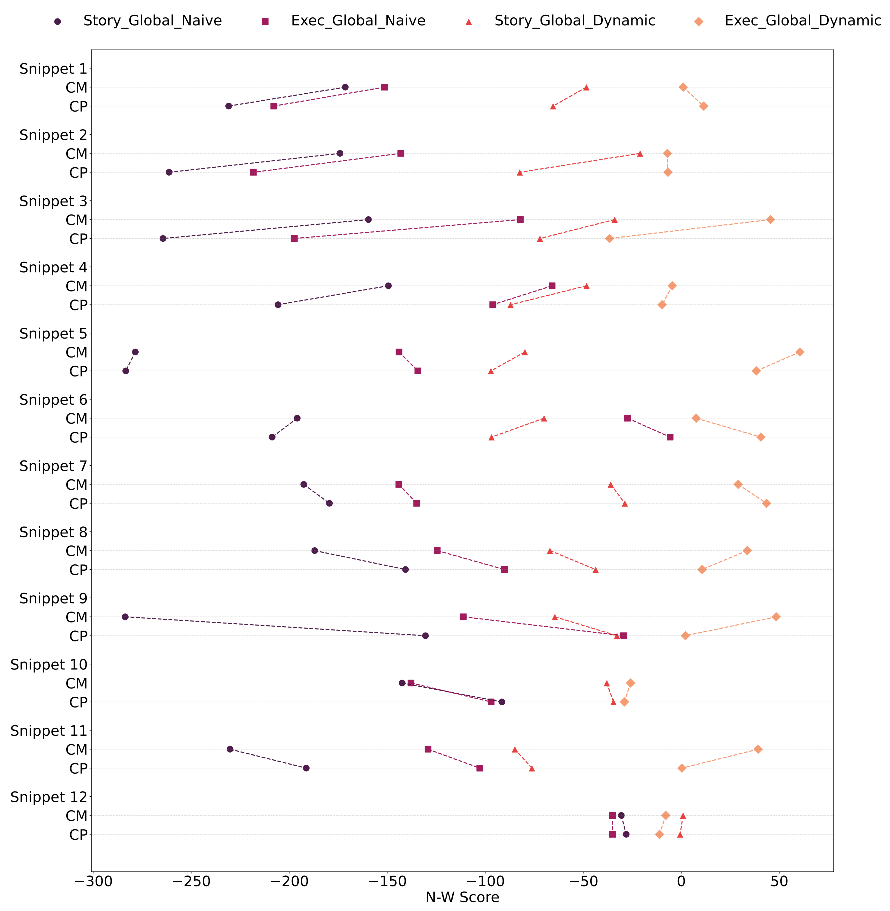


Fig. 9 Global linearity N-W scores for CM and CP snippets. Higher scores translate to better alignment while lower scores indicate poor alignment with the expected reading sequences.

Table 19 LME results for naive linearity metrics of each snippet. Higher numbers are preferable and suggest a better alignment with the compared reading order. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	Story Global Naive				Exec Global Naive			
Snippet	Intercept	Effect of CP	p-Value	Corr. p-Value	Intercept	Effect of CP	p-Value	Corr. p-Value
1	-171.4	-59.5	0.093	0.192	-151.4	-56.5	0.096	0.192
2	-174.1	-87.2	0.001	**<0.001**	-143.1	-75.2	0.001	**<0.001**
3	-159.6	-104.8	0.003	**0.003**	-82.1	-115.3	0.001	**<0.001**
4	-149.4	-56.3	0.001	**<0.001**	-65.9	-30.3	0.290	0.387
5	-278.5	-4.9	0.911	0.911	-144.0	9.6	0.775	0.911
6	-195.9	-12.8	0.762	0.762	-27.4	21.7	0.001	**<0.001**
7	-192.6	13.1	0.593	0.683	-144.1	9.1	0.683	0.683
8	-187.0	46.3	0.237	0.272	-124.5	34.3	0.255	0.272
9	-283.7	153.2	0.001	**<0.001**	-111.2	81.7	0.001	**<0.001**
10	-142.4	50.9	0.210	0.571	-137.9	40.9	0.286	0.571
11	-230.2	38.9	0.585	0.799	-129.2	26.4	0.619	0.799
12	-30.6	2.5	0.800	1.000	-35.1	0.000	1.0	1.0

Table 20 LME results for dynamic linearity metrics of each snippet. Higher numbers are preferable and suggest a better alignment with the compared reading order. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	Story Global Dynamic				Exec Global Dynamic			
Snippet	Intercept	Effect of CP	p-Value	Corr. p-Value	Intercept	Effect of CP	p-Value	Corr. p-Value
1	-48.4	-17.0	0.402	0.402	1.0	10.4	0.181	0.242
2	-21.0	-61.4	<0.001	**<0.001**	-7.1	0.3	0.976	0.976
3	-34.0	-38.1	<0.001	**0.001**	45.5	-82.1	<0.001	**<0.001**
4	-48.3	-38.8	<0.001	**<0.001**	-4.6	-5.2	0.538	0.538
5	-79.8	-17.3	0.518	0.911	60.5	-22.2	0.181	0.723
6	-70.0	-26.8	0.033	0.065	7.6	33.0	0.346	0.461
7	-36.0	7.2	0.506	0.683	29.0	14.5	0.193	0.683
8	-67.0	23.3	0.211	0.272	33.6	-23.0	0.272	0.272
9	-64.5	31.7	<0.001	**<0.001**	48.4	-46.3	0.138	0.138
10	-38.0	3.4	0.829	0.829	-25.9	-3.1	0.675	0.829
11	-84.9	8.7	0.799	0.799	39.2	-38.9	<0.001	**<0.001**
12	0.9	-1.5	0.816	1.000	-7.9	-3.2	0.166	0.663

RQ$_{2.2}$ Our investigation into the influence of comments on the linearity of reading order reveals nuanced interactions. We found that comments significantly alter local reading patterns, leading to a more linear progression through the code. In addition to that, an increase in Horizontal Later Fixations points to more lateral focus within lines, possibly reflecting a deeper engagement with comments. The decrease in Regression Fixations, combined with an uptick in Line Regression Fixations, underscores a more focused yet revisitory reading behavior. Despite these local effects, comments showed limited impact on global reading linearity, when aggregated and measured using the N-W algorithm, challenging our initial hypothesis of a uniform influence. A more detailed analysis of the individual snippets reveals interesting patterns that suggest a more complex relationship than previously anticipated.

6.4 Effect of Comments on Gaze Strategy

Expanding on our prior analysis of reading order linearity, we now explore gaze strategies in programming tasks. For a detailed analysis, we used local and global metrics (cf. *Step IV*c of Section 5.2.3) to provide insights into immediate gaze patterns and reveal how participants navigate between code and accompanying comments on a broader perspective.

6.4.1 Local Metrics

Our findings show that the saccade counts are mostly balanced between the two transition types, with a general tendency for more Code-to-Comment transitions. Certain snippets exhibit statistically significant differences, and overall saccade counts across all snippets are significantly different ($p = 0.007$), indicating programmers transition more from code to comments. Table 21 encapsulates these findings for the local gaze strategy metrics.

Table 21 Comparative analysis of saccade counts between code-to-comment and comment-to-code gaze strategies.

	Local Strategy (Saccade Counts)		Wilcoxon Signed-Rank Test	
Snippet	Code-to-Comment	Comment-to-Code	Statistic	p -Value
1	4.5	4.7	14.5	0.608
2	5.2	4.8	23.5	0.770
3	11.7	8.6	4.0	**0.050**
4	9.5	7.6	4.5	**0.031**
5	13.5	10.3	5.0	**0.020**
6	6.8	6.7	21.0	0.855
7	7.0	7.5	7.5	0.527
8	6.2	5.4	12.0	0.398
9	7.1	5.6	6.5	0.105
10	2.3	1.4	11.0	0.168
11	15.6	12.0	7.0	0.066
12	3.6	2.7	5.0	0.236
Overall	7.8	6.4	6.0	**0.007**

6.4.2 Global Metrics

We now shift our focus to the global metrics. Aligning the participants AOI reading order to the predefined sequences (outlined in Section *Step IV*c) and comparing naive and dynamic scores within both 'Code-First' and 'Comment-First' approaches showed again a consistent pattern of improved performance in the 'Dynamic' calculation further supporting the iterative reading nature of source code. It is however evident that both 'Code-First' and 'Comment-First' approaches result in similar performance scores, with no significant statistical difference. This is demonstrated across different metrics, including Story Order and Execution Order, in both naive and dynamic algorithms. The negative N-W scores across all categories highlight a below-baseline performance, suggesting challenges in aligning gaze patterns even with the best aligned reading sequence. We illustrate the detailed N-W scores of the different reading orders across the snippets in Figure 10. Furthermore, we provide a detailed comparative analysis of the scores between the two gaze strategies in Table 22.

Table 22 Comparative analysis of N-W Scores between code-first and comment-first gaze strategies.

	Global Strategy (N-W Score)		Wilcoxon Signed-Rank Test		
Metric	Code-First	Comment-First	Statistic	p-Value	Corr. p-Value
Story Order Naive	-92.1	-92.0	20.0	0.441	0.441
Execution Order Naive	-65.6	-65.8	21.0	0.284	0.379
Story Order Dynamic	-55.5	-56.5	23.5	0.224	0.379
Execution Order Dynamic	-41.1	-41.9	22.5	0.195	0.379

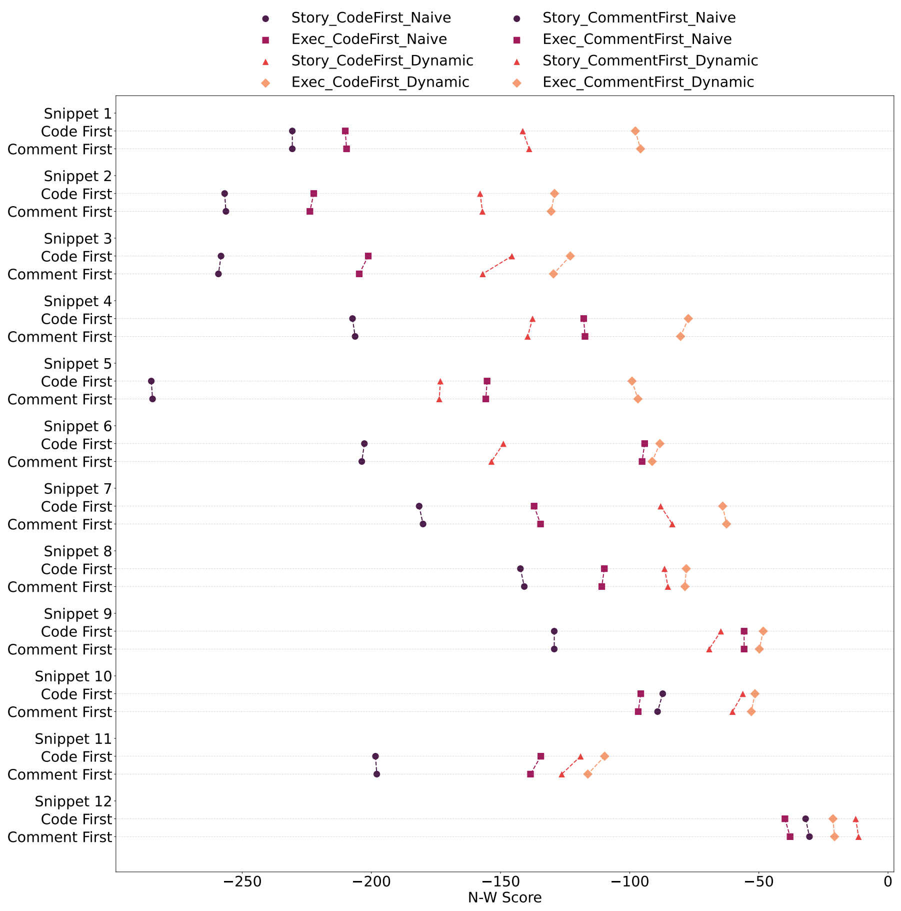


Fig. 10 Gaze strategy N-W scores for CP snippets. Higher scores indicate better alignment while lower scores indicate poor alignment with the expected gaze strategy sequences.

RQ$_{2.3}$ Our findings on gaze strategies highlight a general tendency for programmers to move their gaze from code to comments, indicating a directional strategy that seeks additional context or clarification within comments after having read the code first. Despite this, the assessment of global gaze strategies through the N-W algorithm reveals no significant preference between 'Code-First' and 'Comment-First' approaches. This suggests that, while local gaze transitions favor a move towards comments, the overarching strategy of navigating code and comments does not significantly lean towards starting with one over the other on a global scale.

6.5 Effect of Comments on Participants' Self-Reported Opinions

Finally, we examine participants' difficulty and comment contribution ratings, offering insights into their subjective experiences with program comprehension. This examination is crucial for understanding the perceived impact of comments on navigating code complexity and enhancing the comprehension process.

The examination of difficulty ratings (on a scale of 1 (easy) to 5 (difficult)) across CM and CP snippets uncovers intriguing patterns. For example, Snippet 1 exhibits a slight decrease in perceived difficulty with comments, moving from a mean rating of 2.7 in the CM condition to 2.5 in the CP condition. This suggests a marginal but positive impact of comments. Conversely, Snippet 2 sees an increase in difficulty from 2.4 to 3.1 with the introduction of comments, hinting at the potential for comments to sometimes introduce additional complexity or distraction. Among the different snippets, Snippet 9 exhibited a substantial 34% reduction in perceived difficulty with comments ($p < 0.001$), underscoring comments' potential to significantly clarify and simplify the comprehension process.

Interestingly, the effect of comments on perceived difficulty does not uniformly correlate with their contributory ratings to comprehension. While Snippet 9's significant ease in difficulty is corroborated by a high comment contribution rating (3.9), other snippets with negative or minimal changes in difficulty still report substantial comment contribution scores, such as Snippet 4 (3.7) and Snippet 10 (3.8). This suggests that while comments can directly influence the perceived difficulty of snippets, their value in enhancing comprehension may be perceived independently of this difficulty impact.

Table 23 details the comprehensive results and specific distributions of difficulty and comment contribution ratings across snippets, providing a granular view of how participants' self-reported ratings vary by snippet. Table 24 then synthesizes these individual observations into a broader context, presenting the mean ratings across snippets to summarize the overarching trends in perceived difficulty and the value of comments in program comprehension. Additionally, Figure 11 visualizes these summarized findings and illustrates the relationship between the effect on difficulty and the perceived comment contribution.

Even when comments do not significantly alter the difficulty level, their presence is appreciated for providing context, clarifying intent, or offering insights that aid in understanding complex code segments.

Table 23 Participants' Likert-scale ratings on snippet difficulty and comment contribution.

			Ratings (Likert Scale 1 → 5)	
			Difficulty	Comment Contribution
Snippet	Description	Type	(Very Easy → Very Difficult)	(Not Helpful → Very Helpful)
1	Identifies a peak element in an array.	CM	2 2 3 3	3 7 2 5 3
		CP	2 4 2 1 1	
2	Finds two elements that sum to a target.	CM	2 4 3 1	5 7 3 5
		CP	4 3 1 2	
3	Maximum zero-sum subarray length.	CM	1 2 3 1 3	1 5 4 6 4
		CP	1 1 5 3	
4	Longest consecutive sequence in array.	CM	3 2 5	1 1 6 8 4
		CP	1 4 5	
5	Longest common subsequence between strings.	CM	2 1 1 6	6 5 5 4
		CP	3 2 5	
6	Longest increasing subsequence length.	CM	1 7 2	1 5 1 9 4
		CP	1 4 4 1	
7	Efficient power calculation.	CM	1 5 3 1	6 3 3 4 4
		CP	2 4 2 1 1	
8	Fibonacci number at given position.	CM	4 4 2	4 4 6 5 1
		CP	4 4 2	
9	Sieve of Eratosthenes.	CM	3 4 2 1	7 8 5
		CP	7 2 1	
10	Binary search in sorted array.	CM	4 4 2	1 3 3 6 7
		CP	8 1 1	
11	Counts palindromic substrings.	CM	3 2 3 2	3 4 2 4 7
		CP	2 1 1 3 3	
12	Anagram check for two strings.	CM	8 1 1	7 4 1 2 6
		CP	10	

Table 24 Summary of mean difficulty and comment contribution ratings. The overall effect represents the percentage change in difficulty rating between CM and CP snippets. The cell shading highlights a significant effect of comments (**positive** or **negative**).

	Difficulty (Likert Scale 1 → 5)				Comment
Snippet	CM	CP	Overall Effect	p-Value	Contribution
1	2.7	2.5	-4%	0.603	2.9
2	2.4	3.1	+14%	0.062	3.4
3	3.3	3.0	-6%	0.431	3.4
4	3.2	3.4	+4%	0.564	3.7
5	3.9	4.2	+6%	**<0.001**	3.4
6	4.1	3.5	-12%	0.062	3.5
7	2.4	2.5	+2%	0.769	2.9
8	1.8	1.8	0%	1.000	2.8
9	3.1	1.4	-34%	**<0.001**	3.9
10	1.8	1.3	-10%	0.128	3.8
11	3.4	3.4	0%	1.000	3.4
12	1.4	1.0	-8%	0.168	2.8

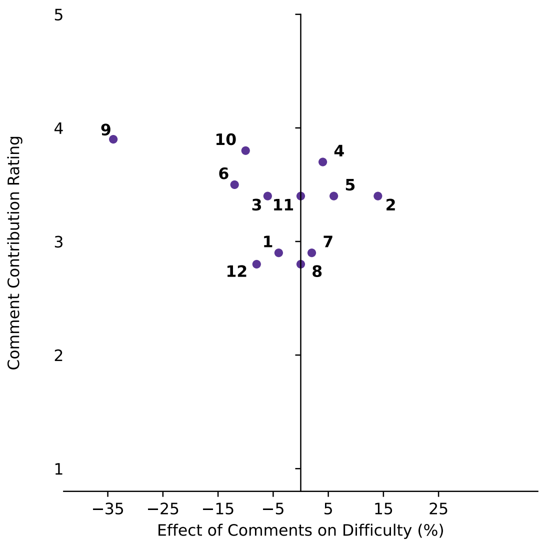


Fig. 11 Effect of comments on difficulty and their contribution ratings.

RQ$_3$ Our analysis of participants' difficulty and comment contribution ratings shows a snippet-dependent influence of comments on perceived difficulty. Notably, Snippet 9 stands out with a substantial 34% reduction in perceived difficulty when comments are present, highlighting the potential of good comments to significantly clarify and simplify the comprehension process. Conversely, other snippets such as Snippet 2 indicate that comments can sometimes introduce additional complexity, suggesting the importance of comment quality and relevance. Despite these variations, consistently high comment contribution ratings across snippets indicate a generally positive perception of comments.

7 Discussion

In this study we aim at exploring the impact of comments on program comprehension, focusing on how comments affect program comprehension performance, visual attention, linearity of reading order, gaze strategies, and participants' self-reported opinions. By conducting a detailed analysis involving eye-tracking data and participant feedback, we have gained insights that contribute to the existing body of knowledge on the role of comments in software development.

Central to our investigation was not only the collection of eye-tracking data but also the gathering of qualitative feedback through a post-questionnaire (cf. Section 4.6) designed to capture participants' self-reported opinions and experiences with comments in code. The incorporation of participant feedback provides a rich, qualitative layer to our analysis, allowing us to dig deeper into the subjective experiences of programmers as they navigate code with and without comments.

The following discussion weaves together the insights from our quantitative analysis with the qualitative themes identified through the thematic analysis (cf. Section 5.4). By integrating these findings, we aim to offer a comprehensive overview of the impact of comments on program comprehension, highlighting the significant role comments play in aiding understanding, guiding attention, and influencing reading and gaze strategies.

7.1 Program Comprehension Performance

Our findings indicate that the impact of comments on program comprehension is complex and varies according to specific contexts, suggesting a more nuanced effect than the uniformly positive improvements reported by Dunsmore [30] and Tenny [114, 115]. For instance, Snippet 9, which listed primes up to a given number, demonstrated a significant improvement in performance when comments were included (+34%). On the other hand, Snippet 1, which identified a peak element in an array, showed a significantly decreased performance as a result of the comments added to it (-30%). This variability in the effectiveness of comments appears to depend on various factors such as the complexity of the code, the quality of the comments, and the background knowledge of the programmer. These findings align with the conditional improvements observed by Woodfield et al. [125] and the varied outcomes for different user groups reported by Salviulo and Scanniello [83]. It suggests that the effectiveness of comments is not universally applicable but contingent on the interplay of various factors, which is also supported by Nurvitadhi et al. [67], who show the importance of considering the specific application and content of comments in programming tasks. In this regard, our study contributes to the discourse on comments in programming by offering evidence of their context-dependent utility, subtly diverging from the findings of Sheppard et al. [95], Börstler and Paech [16], and Nielebock et al. [65], who observed minimal or no influence of comments on comprehension for small tasks.

From the participants' responses several key phrases and concepts emerged, providing perspectives on how comments influence understanding of code. Here are the important reoccurring themes we identified in the responses:

- **Clarification and Summary:** Comments are valued for providing structure and summarizing complex code segments, making them easier to understand. For instance, comments that "give a structure to understanding", or "can summarize complicated code sections simply" are highlighted as particularly helpful.
- **Providing Context and Facilitating Comprehension:** For code snippets that were not immediately clear, comments not only provided necessary context, which was particularly valued in instances of uncertainty, but many responses also indicated that comments facilitated the comprehension process, as seen in phrases like "Comments have generally simplified and sped up my understanding."
- **Intention Behind the Code:** Comments adding new information about the intention behind the code or explaining complex lines are seen as beneficial. However, comments that "merely restate easy to understand lines were not helpful."
- **Relevance and Quality of Comments:** The effectiveness of comments is tied to their relevance and quality. Short and descriptive comments are appreciated for aiding comprehension, whereas "misleading comments" are criticized.
- **Selective Reading Based on Understanding:** Some participants indicated they would not read comments if they understood the code, suggesting a selective approach to using comments based on immediate comprehension needs.

Overall, our analysis reveals that comments play a crucial role in facilitating program comprehension by providing clarification, summarizing complex parts, and explaining the intention behind code. Their utility, however, is highly dependent on their quality and relevance, with effective comments being those that are concise, informative, and directly related to the complexities or uncertainties of the code. The participants' engagement with comments appears to be selective, influenced by their initial understanding of the code and the perceived value of the comments in enhancing that understanding.

Comparing these findings with Nielebock et al. [65], we note some parallels and differences. Like our study, Nielebock et al. found that participants generally viewed comments as potentially helpful in reducing the time for comprehension, which aligns with our observations on speeding up comprehension and providing context. However, they also noted no significant differences in the perceived effectiveness of different types of comments, which contrasts with our findings that suggest that the relevance and quality of comments plays a role in their effectiveness.

In summary, our research adds an additional layer to the understanding of how comments affect program comprehension performance, drawing attention to the subtleties and variations in their impact. Moving forward, these insights pave the way for further research into optimizing the use of comments in programming, focusing on how they can be tailored to enhance comprehension and efficiency in diverse coding scenarios.

7.2 Visual Attention

Our quantitative analysis, enriched by thematic insights from the participants, highlights the significant impact comments have on visual attention during program comprehension. The eye-tracking data demonstrated that a notable portion of visual attention—approx. 23%—was directed towards comments when present, underscoring their vital role in engaging programmers.

This finding is in contrast with that of Karas et al. (2024) [47], who reported substantially lower visual attention to comments during program comprehension tasks. Specifically, Karas et al. observed that comments received about 4.7% of visual attention during reading tasks and only 8.9% during writing tasks, which involved generating code summaries. Given that Karas et al.'s reading task involved evaluating summaries rather than directly engaging with code snippets, it likely required less reliance on comments. Our task, which required a detailed evaluation of code snippets, might be more comparable to their writing task that presumably involved a more thorough understanding of code.

However, even when compared to their writing task, our study shows a substantially higher proportion of attention on comments (23%), indicating a notable difference. This discrepancy could be influenced by several factors including comment density, code complexity, study design, and participant expertise, all of which may have contributed to increased reliance on comments in our task.

Our finding is complemented by the themes identified through participants' feedback, which further elucidate the nuances of how comments attract visual attention:

- *Variable Attention Allocation:* Participants exhibited a wide range of attention distribution between comments and code. While some participants were drawn more to comments, particularly noting variable declarations and output lines, others remained primarily focused on the code, occasionally overlooking comments.
- *Dependency on Code Clarity:* The emphasis on comments was markedly reduced when participants encountered clear code. This suggests that the presence of comments is particularly valued in instances of complexity or when the code lacks intuitiveness, serving as a beacon for clarification.
- *Proportion of Focus:* A striking observation from some participants highlighted a focus distribution where comments commanded a majority of the attention, with ratios self-reported as 65% on comments to 35% on code. This indicates that, under certain circumstances, comments can dominate the visual exploration of code, potentially guiding comprehension and focus.

Furthermore, the thematic analysis uncovered an increased reliance on comments among participants with varying levels of experience, indicating that comments are not just complementary but essential components of code that significantly influence comprehension strategies. Participants frequently shifted their primary focus to comments, particularly in search of clarifications or additional explanations not readily apparent in the code itself. This shift often occurred when comments were perceived as more informative or when the code alone was deemed insufficient for full comprehension.

The relation between comments and visual attention in program comprehension has profound implications for software development practices. Firstly, the findings highlight the importance of clear and informative comments, especially in complex or unclear code segments, to aid programmers in navigating and understanding codebases efficiently. Secondly, the variability in attention allocation between comments and code suggests the need for tailored commenting strategies that consider the diverse preferences and experiences of programmers. Lastly, the significant reliance on comments for understanding emphasizes the value of comments and the need for further investigation on commenting practices.

Notably, the differences between our findings and those of Karas et al. suggest that future research could benefit from more explicit comparisons between studies to better understand how factors such as task design, comment density, and participant expertise influence visual attention patterns. Such work would help clarify the unique contributions of each study and guide more effective commenting strategies to support program comprehension across diverse contexts.

7.3 Linearity of Reading Order

Following our exploration of how comments shape visual attention, we analyzed their influence on the linearity of reading order. Quantitatively, we observed that comments significantly altered local reading patterns, encouraging a more linear, top-to-bottom progression through the code. This was coupled with an increase in Horizontal Later Fixations, suggesting a lateral focus within lines that likely reflects deeper engagement with comments. At the same time, a decrease in Regression Fixations, paired with an increase in Line Regression Fixations, highlighted a more focused yet revisitory reading behavior. These shifts suggest that comments can both guide a systematic approach to reading code and encourage programmers to seek out specific, detailed understanding within the codebase. However, the impact of comments on the global linearity of reading, as measured using the N-W algorithm, was limited when aggregated. This finding challenges the initial hypothesis of a uniform influence of comments on reading order, suggesting a more complex relationship that varies across individual snippets and programmer experiences.

The thematic analysis of participant feedback, sheds light on the different ways comments impact reading strategies:

– *Top-Down vs. Selective Reading:* The presence of comments appears to alter reading strategies from a purely top-down approach to a more selective one. Participants often shifted focus to specific lines of interest, particularly output lines or those directly clarified by comments, indicating a departure from linear navigation to a more targeted engagement with the code.

– *Impact on Code Navigation:* Responses by participants suggest that without comments, programmers might find themselves more inclined to jump around the code. This implies that comments serve not just as explanatory aids but also as navigational beacons, offering a structured pathway through the complexities of code, thereby enhancing the comprehensibility and accessibility of the codebase.
– *Mixed Effects on Linearity:* The effects of comments on the linearity of reading patterns were mixed. While some participants observed no change in their reading order, others noted that comments influenced them to prioritize specific lines. However, a generally top-down reading pattern was maintained, albeit with deviations to accommodate the insights offered by comments.

These insights highlight the significant role that comments play in facilitating program comprehension and navigation, especially in guiding programmers through complex or ambiguous sections. Nonetheless, our study reveals a critical gap in the existing knowledge regarding the optimal placement and formulation of comments within code to maximize their utility. This gap points to the issue that the effectiveness of comments is not universal but may vary significantly among programmers, suggesting that what constitutes a "good" comment is subjective and potentially dependent on individual preferences and coding styles.

This variability in the perceived value of comments emphasizes the need for further exploration into commenting practices and programmer education. It indicates that a universal approach and general guidelines for commenting may not be sufficient. Instead, there is a need for further research to define more effective guidelines that can accommodate the diverse perspectives and preferences of programmers. Specifically, future studies could benefit from exploring the impact of different types of comments on program comprehension by comparing identical code snippets annotated with varied commenting styles.

An intriguing extension of this research could also explore the integration of Large Language Models (LLMs) into Integrated Development Environments (IDEs) to provide personalized comment generation, similar to the work of Wong et al. [124] and McBurney and McMillan [62]. It would be interesting to analyze the code being read and automatically generate comments that are tailored to the programmer's specific reading and comprehension strategies. By doing so, it could significantly reduce the time programmers spend on writing comments and improve code understanding efficiency. Such an approach would not only personalize the development experience but also potentially transform commenting practices by making them more dynamic and adaptive to individual needs.

7.4 Gaze Strategy

Building on our insights into how comments influence visual attention and the linearity of reading order, we studied the comments' impact on gaze strategies during program comprehension. Our investigation aimed to understand how programmers navigate between code and comments.

The findings from our study draw a nuanced picture of how programmers navigate between code and comments, with local metrics indicating a statistically significant preference for a Code-First approach. This preference suggests that, at least in the context of shorter or more focused interactions, programmers tend to engage with the code before seeking out comments for additional context or clarification. In contrast, the global metrics did not reveal a definitive preference for starting with code versus comments, highlighting a potential limitation in our methodology. The application of the N-W algorithm, while effective in certain analytical contexts, may not be ideally suited to capture the slight differences in AOI orders between the sequences, potentially obscuring subtle but significant gaze strategy patterns.

Complementing our quantitative analysis, the thematic analysis of participant feedback offers additional insights into the gaze strategies among programmers. This qualitative approach revealed a clear division in preferences and behaviors:

- *Comment-First vs. Code-First Strategies:* Participants exhibited distinct preferences, with some focusing on comments before delving into the code, while others prioritized understanding the code directly, resorting to comments only when further clarification was needed. A few participants even adopted a "code-only" approach, bypassing comments unless they deemed it absolutely necessary for comprehension.
- *Adaptive Strategies:* A significant insight from our study is the adaptability of programmers' gaze strategies. Participants reported that they adjusted their focus based on the perceived utility of comments, indicating a highly contextual and responsive approach to navigating code. This adaptability suggests that programmers are not rigid in their strategies but are instead capable of dynamically altering their focus to maximize comprehension.

Given these observations, future studies shall consider employing a mix of local and global metrics, potentially integrating alternative measures that are better suited for capturing the intricacies of programmer behavior. An alternative design might involve focusing on shorter code snippets which could offer a more controlled environment to compare code-first versus comment-first reading strategies. Shorter snippets are likely to result in shorter AOI sequences, facilitating a more accurate alignment score with fewer gaps. This methodological adjustment promises a clearer understanding of programmers' navigational preferences by minimizing the confounding factors present in longer code segments.

Overall, the exploration of gaze strategies reveals the critical role comments play as dynamic elements that programmers interact with in a strategic manner. This interaction is not only a matter of preference but is also deeply influenced by the content and perceived utility of comments, suggesting that effective commenting practices are crucial for enhancing program comprehension.

7.5 Participants' Self-Reported Opinions

Following our examination of gaze strategies, we turn our focus towards understanding how participants perceive the role of comments in program comprehension. This perspective offers direct insights into the subjective experience of programmers as they navigate through code, complemented by comments.

Our analysis reveals nuanced patterns that underscore the multifaceted impact of comments. Most notably, Snippet 9 exhibited a notable reduction in difficulty by 34% with comments, highlighting how substantial the effect of well-written comments can be.

Interestingly, the impact of comments on snippet difficulty did not always align with their perceived helpfulness, mirroring the discrepancy found by Nielebock et al. [65] between expectations of comments' utility and actual performance outcomes. Despite participants consistently perceiving and rating comments as helpful, this did not uniformly translate to a reduction in perceived snippet difficulty.

The thematic analysis of participant responses further deepens our understanding, unveiling several key themes:

– *Essential for Understanding Complex Code:* The participants overwhelmingly recognized comments as indispensable, particularly for navigating through challenging code blocks. This highlights comments as "lifelines" in certain situations and indispensable in reducing code complexity.
– *Varied Contribution:* While some participants viewed comments as potentially time-consuming for straightforward code snippets, others praised their role in making conditions, loops, and complex structures more clear.
– *Perception of Code Complexity:* Interestingly, the presence of comments sometimes contributed to an initial perception of increased code complexity. This hints that the mere presence of comments can influence programmers' first impressions, potentially framing their approach to understanding the code.

Combining these insights, our study illustrates that while comments can directly influence the perceived difficulty of programming tasks, they are consistently valued for their broader contribution to comprehension. Even in scenarios where comments appear to increase complexity, their overall contribution to clarifying code logic and intent is highly regarded.

This refined understanding of comments—captured through both quantitative ratings and qualitative themes—highlights their critical role in software development. Comments serve not just as aids for reducing immediate snippet difficulty but as essential tools for enhancing the deeper comprehension of code. They provide context, clarify intent, and offer insights that are crucial for understanding complex code segments.

Our findings suggest that the effectiveness of comments is contingent upon their quality, relevance, and the context in which they are used. Future research should further explore how to optimize comment use to maximize their benefits in program comprehension, considering factors such as code complexity, programmer experience, and the specific needs of different programming tasks and comprehension strategies.

8 Threats To Validity

8.1 Construct Validity

Ensuring construct validity was a key objective in our eye-tracking study to ensure that the measured variables effectively represent the intended constructs. Our study employed various eye-movement measures to assess participants' visual behavior during program comprehension and the interaction between code and comments.

One potential threat was the possibility of participants using peripheral vision or not focusing directly on a source code line during eye-tracking sessions [68]. To address this threat, we employed eye-movement measures that are based on matching fixations to specific AOIs. This approach, similar to the work of Busjahn et al. [21], allowed us to capture participants' actual fixation patterns and align them accurately with the corresponding code elements. By considering horizontal proximity, we accounted for peripheral vision while ensuring a reliable assessment of participants' reading behavior.

Overall, we aimed to ensure the accuracy and reliability of our study's findings by diligently considering and aligning our operationalizations with established measures and methods.

8.2 Internal Validity

Internal validity is crucial to ensure the accurate identification of the cause-and-effect relationships between the independent and dependent variables in our study. To enhance internal validity, we have employed a carefully designed within-subject experimental design. Each participant was exposed to both conditions (Comments Present and Comments Missing) in a counterbalanced order to control for order effects. The random assignment of code snippets to participants further minimized potential biases or confounding factors.

The use of eye-tracking data allowed for accurate and objective measurements of participants' visual attention during program comprehension. To ensure the reliability of eye-tracking data, the eye tracker was carefully positioned according to the manufacturer's instructions, and participants underwent a calibration process before the study.

To minimize potential biases, the study was conducted in a controlled environment. Each snippet was presented only once in a random condition to avoid any potential learning effects. Participants received clear written instructions and a warm-up snippet to ensure they are familiar with the study procedures, reducing the risk of misunderstanding or confusion.

Furthermore, the selected code snippets have undergone careful adaptations to standardize tasks, remove code documentation, and obfuscate obvious function and variable names. These modifications aimed at minimizing potential biases arising from specific coding styles or identifier names and allowed us to focus on the influence of comments on comprehension.

By employing established measures and methodologies, our study ensured accurate representations of program comprehension performance, visual attention, linearity, and gaze strategies during code and comment interaction. The rigorous experimental design and the control of potential confounding factors enhance the internal validity of our findings.

8.3 External and Ecological Validity

External validity concerns how well our study's findings can be applied to different populations, settings, and contexts beyond our specific sample and conditions.

Generally, when conducting empirical studies, we must consider the trade-off between enhancing internal or external validity [99]. Our goal for this study was to robustly pinpoint the effects of comments on program comprehension and, thus, we optimized our study for internal validity. While our study provides valuable insights into the effect of comments, it is essential to consider certain limitations that may affect the generalizability of the results.

The sample for our study consists of computer science students with some programming experience, which may not fully represent professional software developers or individuals from different academic backgrounds. Therefore, the generalizability of our findings to a broader population of programmers should be interpreted with caution.

Furthermore, the participants were on a range of expertise levels, which could have influenced gaze patterns and cognitive processing during tasks. More experienced programmers might exhibit more efficient eye movement patterns, shorter fixations on certain code elements, less reliance on comments, or different transition behaviors compared to less experienced participants. However, due to the sample size and study design, we did not conduct a formal subgroup analysis based on expertise levels.

Additionally, the study focused on relatively simple code snippets with approximately 20 lines of code. While this design choice allows us to investigate the role of comments in understanding basic code components, the findings may not fully capture the effects of comments in more complex codebases or domains. The use of a controlled environment and code snippets also differs from the dynamics of real-world software development, potentially affecting the ecological validity.

Future research shall consider expanding the participant pool to include programmers with diverse levels of experience and backgrounds across different programming domains, enabling a more nuanced investigation into how varying expertise influences eye-tracking metrics and comprehension strategies. Furthermore, studying larger and more complex codebases could provide insights into how comments influence program comprehension in real-world scenarios.

9 Conclusion

This paper set out to explore the impact of comments on program comprehension among computer science students. By employing a mixed-methods approach, we combined comprehension tasks to collect behavioral data, eye-tracking analysis to capture visual attention patterns, and a post-questionnaire to gather subjective opinions from participants. We presented Java code snippets both with and without comments, and evaluated the participants' performance on the correctness and response time of their comprehension tasks. Additionally, we recorded their eye gaze to analyze fixation patterns and reading order. By providing a quantifiable analysis of how comments impact various aspects of program comprehension, this research marks a notable contribution to the field, offering empirical evidence to deepen our understanding of the role of comments in software development. Our findings revealed a complex relation between comments and the various dimensions of program comprehension investigated, offering nuanced insights as outlined below:

RQ$_1$: Program Comprehension Performance: Our study reveals that comments have varying impact on program comprehension performance. Some code snippets showed improved performance with comments, indicating their potential to enhance understanding, while others showed a decrease in performance, highlighting the context-dependent effectiveness of comments. The feedback of participants supported these findings, noting that the value of comments lies in their ability to clarify and summarize complex code segments, offer context, and elucidate code intentions. This suggests that the efficacy of comments in programming is highly reliant on their relevance, quality, and the specific context of the code.

RQ$_2$: Reading Behavior: The presence of comments in code significantly influences reading behavior. Comments not only shift visual attention and promote more structured reading patterns but also shape the gaze strategies employed by programmers.

RQ$_{2.1}$: Visual Attention: We found that comments notably shift visual attention during code comprehension, with approximately 23% of focus redirected from code to comments. This demonstrates the comments' critical role in influencing how programmers engage with and understand code. Participant feedback supported these findings, indicating that comments are particularly valuable in navigating complex or unclear code segments.

RQ$_{2.2}$: Linearity of Reading Order: The study found that comments influence the linearity of reading order, encouraging a more linear, top-to-bottom reading pattern locally. Globally, the impact of comments on reading linearity was snippet-dependent, suggesting a complex interplay between comments and reading strategies. Participant feedback also revealed that comments can alter reading strategies from a purely top-down approach to a more selective reading order, where programmers targeted specific lines of interest clarified by comments. These findings indicate that comments serve as navigational aids in code comprehension, particularly in guiding through complex sections, and underscore the importance of effective commenting practices to enhance program understanding.

RQ$_{2.3}$: Gaze Strategies: The investigation into gaze strategies revealed a significant tendency for programmers to shift their gaze from code to comments, suggesting a directional strategy that integrates additional context or clarifications found within comments. The thematic analysis further highlighted that participants adapted their gaze strategies based on the perceived utility of comments. This indicates the significant yet variable role of comments in programming, emphasizing the need for flexible commenting practices that accommodate diverse programming tasks and individual preferences.

RQ$_3$: Participants' Self-Reported Opinions: The study on participants' self-reported opinions highlighted that comments can both simplify and complicate program comprehension, with significant variability across tasks. Notably, comments significantly reduced perceived difficulty in certain tasks, underscoring their potential to enhance comprehension significantly. The thematic analysis additionally revealed that while comments were indispensable in understanding challenging code blocks, their mere presence sometimes contributed to an initial perception of increased code complexity. Despite this, comments were universally valued for their contribution to understanding, suggesting that their role extends beyond simplifying tasks to enriching overall comprehension.

These findings contribute to a deeper understanding of the role of comments in program comprehension, suggesting that, while comments can be a powerful tool for enhancing program comprehension, their impact is not universally positive nor negative and depends on various factors including the complexity of the code, the quality and relevance of the comments, and the individual programmer's strategy for navigating code and comments.

The implications of these insights extend beyond academic inquiry into potential practical applications for integrating adaptive commenting systems within development environments. Such systems could leverage insights from our study to dynamically adjust the presentation or emphasis of comments based on the programmer's current task, their reading patterns, and possibly even their historical interaction with similar code structures. This personalized approach could optimize program comprehension by ensuring that comments serve as effective guides through the code, enhancing understanding without overwhelming the programmer with unnecessary information.

In conclusion, this paper not only contributes to the theoretical understanding of the role of comments in program comprehension but also opens avenues for practical applications and future research. By shedding light on how comments influence various aspects of program comprehension, we offer a foundation for developing more effective commenting practices and tools that can adapt to the needs of individual programmers. Our results point towards the need for further research into the cognitive processes underlying programming. Understanding the interplay between comments and program comprehension could provide a foundation for designing more intuitive and supportive development tools and environments.

Future work in this area could focus on developing algorithms or models that predict the usefulness of comments based on the code context, programmer expertise, and task complexity. Additionally, exploring the role of automated comment generation and its effectiveness compared to human-written comments could offer insights into how best to support programmers in their work. Ultimately, this paper lays the groundwork for a more nuanced understanding of comments in programming, advocating for their strategic use to improve program comprehension and the software development process at large.

Data Availability

We provide the raw data and analysis scripts in our replication package.

Conflict of Interest

The authors declare that they have no conflict of interest.

Ethical Approval

Our study did not need to be approved by the ethics committee of our host institution (Saarland University) as it was a student thesis project. Nevertheless, the experiment was conducted according to the relevant guidelines and regulations provided by the ethical review board.

Informed Consent

All participants provided their informed consent. They did not receive any compensation.

Author Contributions

Youssef Abdelsalam, Norman Peitek, Annabelle Bergum, and Sven Apel developed the concept and the general research idea. They also devised the study design and formulated the method. Youssef Abdelsalam conducted the study, performed the data analysis, and prepared the visualizations. All authors contributed to the interpretation, writing, editing and reviewing. Sven Apel provided the funding and supervised the project.

Acknowledgements We thank all participants of our study. This work has been supported by ERC Advanced Grant 101052182 as well as DFG Grant 389792660 as part of TRR 248 – CPEC.

References

1. Adeli, M., Nelson, N., Chattopadhyay, S., Coffey, H., Henley, A., Sarma, A.: Supporting Code Comprehension via Annotations: Right Information at the Right Time and Place. In: 2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). IEEE (2020). DOI 10.1109/vl/hcc50065.2020.9127264
2. Ali, N., Sharafi, Z., Guéhéneuc, Y.G., Antoniol, G.: An Empirical Study on the Importance of Source Code Entities for Requirements Traceability. Empirical Software Engineering **20**(2), 442–478 (2014). DOI 10.1007/s10664-014-9315-y
3. Antinyan, V.: Evaluating Essential and Accidental Code Complexity Triggers by Practitioners' Perception. IEEE Software **37**(6), 86–93 (2020). DOI 10.1109/MS.2020.2976072
4. Antoniol, G., Canfora, G., Casazza, G., Lucia, A.D., Merlo, E.: Recovering Traceability Links between Code and Documentation. IEEE Transactions on Software Engineering **28**(10), 970–983 (2002). DOI 10.1109/tse.2002.1041053
5. Baddeley, A.D., Hitch, G.: Working Memory. In: G.H. Bower (ed.) Psychology of Learning and Motivation, vol. 8, pp. 47–89. Academic Press (1974). DOI https://doi.org/10.1016/S0079-7421(08)60452-1. URL `https://www.sciencedirect.com/science/article/pii/S0079742108604521`
6. Basili, V.R., Shull, F., Lanubile, F.: Building Knowledge through Families of Experiments. IEEE Transactions on Software Engineering **25**(4), 456–473 (1999). DOI 10.1109/32.799939
7. Bednarik, R.: Methods to Analyze Visual Attention Strategies: Applications in the Studies of Programming. University of Joensuu (2007)
8. Bednarik, R., Eivazi, S., Hradis, M.: Gaze and Conversational Engagement in Multiparty Video Conversation. In: Proceedings of the 4th Workshop on Eye Gaze in Intelligent Human Machine Interaction. ACM (2012). DOI 10.1145/2401836.2401846
9. Beelders, T.: Eye-Tracking Analysis of Source Code Reading on a Line-By-Line Basis. In: Proceedings of the 10th International Workshop on Eye Movements in Programming. ACM (2022). DOI 10.1145/3524488.3527364
10. Benjamini, Y., Hochberg, Y.: Controlling The False Discovery Rate - A Practical And Powerful Approach To Multiple Testing. J. Royal Statist. Soc., Series B **57**, 289–300 (1995). DOI 10.2307/2346101

11. Berón, M., Rangel Henriques, P., Pereira, M., Uzal, R.: Program Inspection to Interconnect Behavioral and Operational View for Program Comprehension. In: ESTiG - Artigos em Proceedings Não Indexados à WoS/Scopus. University of York (2007)
12. Bezerra, R.M.M., da Silva, F.Q.B., Santana, A.M., Magalhaes, C.V.C., Santos, R.E.S.: Replication of Empirical Studies in Software Engineering: An Update of a Systematic Mapping Study. In: 2015 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM). IEEE (2015). DOI 10.1109/esem.2015.7321213
13. Binkley, D., Davis, M., Lawrie, D., Maletic, J.I., Morrell, C., Sharif, B.: The Impact of Identifier Style on Effort and Comprehension. Empirical Software Engineering **18**(2), 219–276 (2012). DOI 10.1007/s10664-012-9201-4
14. Blascheck, T., Schweizer, M., Beck, F., Ertl, T.: Visual Comparison of Eye Movement Patterns. Computer Graphics Forum **36**(3), 87–97 (2017). DOI 10.1111/cgf.13170
15. Blinman, S., Cockburn, A.: Program Comprehension: Investigating the Effects of Naming Style and Documentation. In: Proceedings of the 6th Australasian Conference on User Interface - Volume 40, AUIC '05, pp. 73–78. Australian Computer Society, Inc., AUS (2005)
16. Börstler, J., Paech, B.: The Role of Method Chains and Comments in Software Readability and Comprehension—an Experiment. IEEE Transactions on Software Engineering **42**(9), 886–898 (2016). DOI 10.1109/tse.2016.2527791
17. Brooks, F.P.: No Silver Bullet Essence and Accidents of Software Engineering. Computer **20**(4), 10–19 (1987). DOI 10.1109/MC.1987.1663532
18. Brooks, R.: Using a Behavioral Theory of Program Comprehension in Software Engineering. In: Proceedings of the 3rd International Conference on Software Engineering, ICSE '78, pp. 196–201. IEEE Press, Atlanta, Georgia, USA (1978)
19. Brooks, R.: Towards a Theory of the Comprehension of Computer Programs. International Journal of Man-Machine Studies **18**(6), 543–554 (1983). DOI 10.1016/s0020-7373(83)80031-5
20. Buse, R.P.L., Weimer, W.R.: Learning a Metric for Code Readability. IEEE Transactions on Software Engineering **36**(4), 546–558 (2010). DOI 10.1109/tse.2009.70
21. Busjahn, T., Bednarik, R., Begel, A., Crosby, M., Paterson, J.H., Schulte, C., Sharif, B., Tamm, S.: Eye Movements in Code Reading: Relaxing the Linear Order. In: 2015 IEEE 23rd International Conference on Program Comprehension. IEEE (2015). DOI 10.1109/icpc.2015.36
22. Busjahn, T., Schulte, C., Busjahn, A.: Analysis of Code Reading to Gain More Insight in Program Comprehension. In: Proceedings of the 11th Koli Calling International Conference on Computing Education Research. ACM (2011). DOI 10.1145/2094131.2094133
23. Chen, F., Zhou, J., Wang, Y., Yu, K., Arshad, S.Z., Khawaji, A., Conway, D.: Robust Multimodal Cognitive Load Measurement. Springer (2016)
24. Corazza, A., Maggio, V., Scanniello, G.: On the Coherence between Comments and Implementations in Source Code. In: 2015 41st Euromicro Conference on Software Engineering and Advanced Applications. IEEE (2015). DOI 10.1109/seaa.2015.20
25. Corbi, T.A.: Program Understanding: Challenge for the 1990s. IBM Systems Journal **28**(2), 294–306 (1989). DOI 10.1147/sj.282.0294
26. Crosby, M., Scholtz, J., Wiedenbeck, S.: The Roles Beacons Play in Comprehension for Novice and Expert Programmers. In: Annual Workshop of the Psychology of Programming Interest Group (2002)
27. Crosby, M.E., Stelovsky, J.: How Do We Read Algorithms? A Case Study. Computer **23**(1), 24–35 (1990)
28. Duchowski, A.: Eye Tracking Methodology: Theory and Practice. Springer London (2007). DOI 10.1007/978-1-84628-609-4
29. Dunsmore, A., Roper, M., Wood, M.: The Role of Comprehension in Software Inspection. Journal of Systems and Software **52**(2-3), 121–129 (2000). DOI 10.1016/s0164-1212(99)00138-7
30. Dunsmore, H.E.: The Effect of Comments, Mnemonic Names, and Modularity: Some University Experiment Results. Empirical Foundations of Information and Software Science pp. 189–196 (1985)
31. Elshoff, J.L., Marcotty, M.: Improving Computer Program Readability to Aid Modification. Communications of the ACM **25**(8), 512–521 (1982). DOI 10.1145/358589.358596
32. Etzkorn, L.H., Davis, C.G., Bowen, L.L.: The Language of Comments in Computer Software: A Sublanguage of English. Journal of Pragmatics **33**(11), 1731–1756 (2001). DOI 10.1016/s0378-2166(00)00068-0

33. Fakhoury, S., Ma, Y., Arnaoudova, V., Adesope, O.: The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load. In: Proceedings of the 26th Conference on Program Comprehension. ACM (2018). DOI 10.1145/3196321.3196347
34. Fjeldstad, R.K.: Application Program Maintenance Study. Report to Our Respondents, Proceedings GUIDE **48** (1983)
35. Fluri, B., Wursch, M., Gall, H.C.: Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In: 14th Working Conference on Reverse Engineering (WCRE 2007). IEEE (2007). DOI 10.1109/wcre.2007.21
36. Fritz, T., Begel, A., Müller, S.C., Yigit-Elliott, S., Züger, M.: Using Psycho-Physiological Measures to Assess Task Difficulty in Software Development. In: Proceedings of the 36th International Conference on Software Engineering. ACM (2014). DOI 10.1145/2568225.2568266
37. Goldberg, J.H., Kotval, X.P.: Computer Interface Evaluation Using Eye Movements: Methods and Constructs. International Journal of Industrial Ergonomics **24**(6), 631–645 (1999). DOI 10.1016/s0169-8141(98)00068-7
38. Guéhéneuc, Y.G.: TAUPE: Towards Understanding Program Comprehension. In: Proceedings of the 2006 Conference of the Center for Advanced Studies on Collaborative Research - CASCON '06. ACM Press (2006). DOI 10.1145/1188966.1188968
39. Haiduc, S., Marcus, A.: On the Use of Domain Terms in Source Code. In: 2008 16th IEEE International Conference on Program Comprehension. IEEE (2008). DOI 10.1109/icpc.2008.29
40. Hart, S.G., Staveland, L.E.: Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In: P.A. Hancock, N. Meshkati (eds.) Human Mental Workload, *Advances in Psychology*, vol. 52, pp. 139–183. North-Holland (1988). DOI https://doi.org/10.1016/S0166-4115(08)62386-9. URL `https://www.sciencedirect.com/science/article/pii/S0166411508623869`
41. Holmqvist, K., Nyström, M., Mulvey, F.: Eye Tracker Data Quality. In: Proceedings of the Symposium on Eye Tracking Research and Applications. ACM (2012). DOI 10.1145/2168556.2168563
42. Jacob, R.J.K., Karn, K.S.: Eye Tracking in Human-Computer Interaction and Usability Research. In: The Mind's Eye, pp. 573–605. Elsevier (2003). DOI 10.1016/b978-044451020-4/50031-1
43. Ji, W., Berger, T., Antkiewicz, M., Czarnecki, K.: Maintaining Feature Traceability with Embedded Annotations. In: Proceedings of the 19th International Conference on Software Product Line. ACM (2015). DOI 10.1145/2791060.2791107
44. Jiang, Z.M., Hassan, A.E.: Examining the Evolution of Code Comments in PostgreSQL. In: Proceedings of the 2006 International Workshop on Mining Software Repositories. ACM (2006). DOI 10.1145/1137983.1138030
45. Juristo, N., Vegas, S.: Using Differences among Replications of Software Engineering Experiments to Gain Knowledge. In: 2009 3rd International Symposium on Empirical Software Engineering and Measurement. IEEE (2009). DOI 10.1109/esem.2009.5314236
46. Just, M.A., Carpenter, P.A.: A Theory of Reading: From Eye Fixations to Comprehension. Psychological Review **87**(4), 329–354 (1980). DOI 10.1037/0033-295x.87.4.329
47. Karas, Z., Bansal, A., Zhang, Y., Li, T., McMillan, C., Huang, Y.: A Tale of Two Comprehensions? Analyzing Student Programmer Attention during Code Summarization. ACM Trans. Softw. Eng. Methodol. **33**(7) (2024). DOI 10.1145/3664808. URL `https://doi.org/10.1145/3664808`
48. Kernighan, B.W.: The Elements of Programming Style. McGraw-Hill (1974)
49. Knuth, D.E.: Literate Programming. The Computer Journal **27**(2), 97–111 (1984). DOI 10.1093/comjnl/27.2.97
50. Koschke, R., Marcus, A., Gannod, G.C.: Guest Editor's Introduction to the Special Section on the 2009 International Conference on Program Comprehension (ICPC 2009). Software Quality Journal **19**(1), 3–4 (2010). DOI 10.1007/s11219-010-9119-2
51. Krüger, J., Gu, W., Shen, H., Mukelabai, M., Hebig, R., Berger, T.: Towards a Better Understanding of Software Features and Their Characteristics. In: Proceedings of the 12th International Workshop on Variability Modelling of Software-Intensive Systems. ACM (2018). DOI 10.1145/3168365.3168371
52. Lee, S., Hooshyar, D., Ji, H., Nam, K., Lim, H.: Mining Biometric Data to Predict Programmer Expertise and Task Difficulty. Cluster Computing **21**(1), 1097–1107 (2017). DOI 10.1007/s10586-017-0746-2

53. Lehman, M.M., Parr, F.N.: Program Evolution and Its Impact on Software Engineering. In: Proceedings of the 2nd International Conference on Software Engineering, ICSE '76, pp. 350–357. IEEE Computer Society Press, Washington, DC, USA (1976)
54. Letovsky, S.: Cognitive Processes in Program Comprehension. Journal of Systems and Software **7**(4), 325–339 (1987). DOI 10.1016/0164-1212(87)90032-x
55. Likert, R.: A Technique for the Measurement of Attitudes. Archives of Psychology **22**(140), 1–55 (1932)
56. Littman, D.C., Pinto, J., Letovsky, S., Soloway, E.: Mental Models and Software Maintenance. Journal of Systems and Software **7**(4), 341–355 (1987). DOI 10.1016/0164-1212(87)90033-1
57. Luís, J., de Freitas, F.: Comment Analysis for Program Comprehension. Ph.D. thesis, Universidade do Minho (Portugal) (2011)
58. Maguire, M., Delahunt, B.: Doing a thematic analysis: A practical, step-by-step guide for learning and teaching scholars. All Ireland Journal of Higher Education **9**(3) (2017)
59. von Mayrhauser, A., Vans, A.M.: Comprehension Processes during Large Scale Maintenance. In: Proceedings of the 16th International Conference on Software Engineering, ICSE '94, pp. 39–48. IEEE Computer Society Press, Washington, DC, USA (1994)
60. Mayrhauser, A.V., Vans, A.M.: Program Comprehension during Software Maintenance and Evolution. Computer **28**(8), 44–55 (1995). DOI 10.1109/2.402076
61. McBurney, P.W., McMillan, C.: An Empirical Study of the Textual Similarity between Source Code and Source Code Summaries. Empirical Software Engineering **21**(1), 17–42 (2014). DOI 10.1007/s10664-014-9344-6
62. McBurney, P.W., McMillan, C.: Automatic Documentation Generation Via Source Code Summarization of Method Context. In: Proceedings of the 22nd International Conference on Program Comprehension. ACM (2014). DOI 10.1145/2597008.2597149
63. Miller, G.A.: The magical number seven, plus or minus two: Some limits on our capacity for processing information. Psychological review **63**(2), 81 (1956)
64. Nevalainen, S., Sajaniemi, J.: Comparison of Three Eye Tracking Devices in Psychology of Programming Research. In: Annual Workshop of the Psychology of Programming Interest Group (2004)
65. Nielebock, S., Krolikowski, D., Krüger, J., Leich, T., Ortmeier, F.: Commenting Source Code: Is It Worth It for Small Programming Tasks? Empirical Software Engineering **24**(3), 1418–1457 (2018). DOI 10.1007/s10664-018-9664-z
66. Norcio, A.F.: Indentation, Documentation and Programmer Comprehension. In: Proceedings of the 1982 conference on Human factors in computing systems - CHI '82. ACM Press (1982). DOI 10.1145/800049.801766
67. Nurvitadhi, E., Leung, W., Cook, C.: Do Class Comments Aid Java Program Understanding? In: 33rd Annual Frontiers in Education, 2003. FIE 2003., vol. 1, pp. T3C–13. IEEE (2003). DOI 10.1109/fie.2003.1263332
68. Orlov, P.A.: Ambient and Focal Attention during Source-Code Comprehension. FACHBEREICH MATHEMATIK UND INFORMATIK SERIE B INFORMATIK p. 12 (2017)
69. Paas, F., Tuovinen, J., Tabbers, H., Van Gerven, P.: Cognitive Load Measurement as a Means to Advance Cognitive Load Theory. Educational Psychologist - EDUC PSYCHOL **38**, 63–71 (2003). DOI 10.1207/S15326985EP3801_8
70. Padioleau, Y., Tan, L., Zhou, Y.: Listening to Programmers Taxonomies and Characteristics of Comments in Operating System Code. In: 2009 IEEE 31st International Conference on Software Engineering. IEEE (2009). DOI 10.1109/icse.2009.5070533
71. Parkin, P.: An Exploratory Study of Code and Document Interactions during Task-Directed Program Comprehension. In: 2004 Australian Software Engineering Conference. Proceedings. IEEE (2004). DOI 10.1109/aswec.2004.1290475
72. Peitek, N., Apel, S., Parnin, C., Brechmann, A., Siegmund, J.: Program Comprehension and Code Complexity Metrics: An fMRI Study. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 524–536 (2021). DOI 10.1109/ICSE43902.2021.00056
73. Peitek, N., Siegmund, J., Apel, S.: What Drives the Reading Order of Programmers? In: Proceedings of the 28th International Conference on Program Comprehension. ACM (2020). DOI 10.1145/3387904.3389279
74. Peitek, N., Siegmund, J., Apel, S., Kastner, C., Parnin, C., Bethmann, A., Leich, T., Saake, G., Brechmann, A.: A Look into Programmers' Heads. IEEE Transactions on Software Engineering **46**(4), 442–462 (2020). DOI 10.1109/tse.2018.2863303

75. Pennington, N.: Comprehension Strategies in Programming, pp. 100–113. Ablex Publishing Corp., USA (1987)
76. Poole, A., Ball, L.: Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future Prospects, pp. 211–219. IGI Global (2006). DOI 10.4018/978-1-59140-562-7.ch034
77. Privitera, C.M., Stark, L.W.: Algorithms for Defining Visual Regions-Of-Interest: Comparison with Eye Fixations. IEEE Transactions on Pattern Analysis and Machine Intelligence **22**(9), 970–982 (2000). DOI 10.1109/34.877520
78. Ratol, I.K., Robillard, M.P.: Detecting Fragile Comments. In: 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE (2017). DOI 10.1109/ase.2017.8115624
79. Rayner, K.: Eye Movements in Reading and Information Processing. Psychological Bulletin **85**(3), 618–660 (1978). DOI 10.1037/0033-2909.85.3.618
80. Rouam, S.: False Discovery Rate (FDR), pp. 731–732. Springer New York, New York, NY (2013). DOI 10.1007/978-1-4419-9863-7_223
81. Rugaber, S.: Program Comprehension. Encyclopedia of Computer Science and Technology **35**(20), 341–368 (1995)
82. Rugaber, S.: The Use of Domain Knowledge in Program Understanding. Annals of Software Engineering **9**(1/4), 143–192 (2000). DOI 10.1023/a:1018976708691
83. Salviulo, F., Scanniello, G.: Dealing with Identifiers and Comments in Source Code Comprehension and Maintenance. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering. ACM (2014). DOI 10.1145/2601248.2601251
84. Salvucci, D.D., Goldberg, J.H.: Identifying Fixations and Saccades in Eye-Tracking Protocols. In: Proceedings of the Symposium on Eye Tracking Research and Applications. ACM Press (2000). DOI 10.1145/355017.355028
85. Seaman, C.B.: Software Maintenance: Concepts and Practice Authored by Penny Grubb and Armstrong A. Takang World Scientific, New Jersey. Journal of Software Maintenance and Evolution: Research and Practice **20**(6), 463–466 (2008). DOI 10.1002/smr.365
86. Seiler, M., Paech, B.: Using Tags to Support Feature Management across Issue Tracking Systems and Version Control Systems. In: Requirements Engineering: Foundation for Software Quality, pp. 174–180. Springer International Publishing (2017). DOI 10.1007/978-3-319-54045-0_13
87. Shaft, T.M., Vessey, I.: The Relevance of Application Domain Knowledge: Characterizing the Computer Program Comprehension Process. Journal of Management Information Systems **15**(1), 51–78 (1998). DOI 10.1080/07421222.1998.11518196
88. Sharafi, Z., Bertram, I., Flanagan, M., Weimer, W.: Eyes on Code: A Study on Developers' Code Navigation Strategies. IEEE Transactions on Software Engineering **48**(5), 1692–1704 (2022). DOI 10.1109/TSE.2020.3032064
89. Sharafi, Z., Shaffer, T., Sharif, B., Guéhéneuc, Y.G.: Eye-Tracking Metrics in Software Engineering. In: 2015 Asia-Pacific Software Engineering Conference (APSEC). IEEE (2015). DOI 10.1109/apsec.2015.53
90. Sharafi, Z., Sharif, B., Guéhéneuc, Y.G., Begel, A., Bednarik, R., Crosby, M.: A Practical Guide on Conducting Eye Tracking Studies in Software Engineering. Empirical Software Engineering **25**(5), 3128–3174 (2020). DOI 10.1007/s10664-020-09829-4
91. Sharafi, Z., Soh, Z., Guéhéneuc, Y.G.: A Systematic Literature Review on the Usage of Eye-Tracking in Software Engineering. Information and Software Technology **67**, 79–107 (2015). DOI 10.1016/j.infsof.2015.06.008
92. Sharif, B., Falcone, M., Maletic, J.I.: An Eye-Tracking Study on the Role of Scan Time in Finding Source Code Defects. In: Proceedings of the Symposium on Eye Tracking Research and Applications. ACM (2012). DOI 10.1145/2168556.2168642
93. Sharif, B., Jetty, G., Aponte, J., Parra, E.: An Empirical Study Assessing the Effect of Seeit 3d on Comprehension. In: 2013 First IEEE Working Conference on Software Visualization (VISSOFT). IEEE (2013). DOI 10.1109/vissoft.2013.6650519
94. Sharif, B., Maletic, J.I.: An Eye Tracking Study on camelCase and under_score Identifier Styles. In: 2010 IEEE 18th International Conference on Program Comprehension. IEEE (2010). DOI 10.1109/icpc.2010.41
95. Sheppard, S.B., Curtis, B.: Predicting Programmers' Ability to Modify Software. General Electric Company, DTIC Document# TR pp. 78–388100 (1978)
96. Shinyama, Y., Arahori, Y., Gondow, K.: Analyzing Code Comments to Boost Program Comprehension. In: 2018 25th Asia-Pacific Software Engineering Conference (APSEC). IEEE (2018). DOI 10.1109/apsec.2018.00047

97. Shneiderman, B., Mayer, R.: Syntactic/Semantic Interactions in Programmer Behavior: A Model and Experimental Results. International Journal of Computer $&$ Information Sciences **8**(3), 219–238 (1979). DOI 10.1007/bf00977789
98. Siegmund, J.: Program Comprehension: Past, Present, and Future. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER). IEEE (2016). DOI 10.1109/saner.2016.35
99. Siegmund, J., Siegmund, N., Apel, S.: Views on Internal and External Validity in Empirical Software Engineering. In: Proceedings of the International Conference on Software Engineering, vol. 1, pp. 9–19. IEEE (2015)
100. Soh, Z., Khomh, F., Guéhéneuc, Y.G., Antoniol, G.: Noise in Mylyn Interaction Traces and Its Impact on Developers and Recommendation Systems. Empirical Software Engineering **23**(2), 645–692 (2017). DOI 10.1007/s10664-017-9529-x
101. Soh, Z., Khomh, F., Guéhéneuc, Y.G., Antoniol, G., Adams, B.: On the Effect of Program Exploration on Maintenance Tasks. In: 2013 20th Working Conference on Reverse Engineering (WCRE). IEEE (2013). DOI 10.1109/wcre.2013.6671314
102. Soloway, E., Adelson, B., Ehrlich, K.: Knowledge and Processes in the Comprehension of Computer Programs. The nature of expertise pp. 129–152 (1988)
103. Soloway, E., Ehrlich, K.: Empirical Studies of Programming Knowledge. IEEE Transactions on Software Engineering **SE-10**(5), 595–609 (1984). DOI 10.1109/tse.1984.5010283
104. Sommerlad, P., Zgraggen, G., Corbat, T., Felber, L.: Retaining Comments When Refactoring Code. In: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. ACM (2008). DOI 10.1145/1449814.1449817
105. Sorg, T., Abbad-Andaloussi, A., Weber, B.: Towards a Fine-grained Analysis of Cognitive Load During Program Comprehension. In: 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 748–752 (2022). DOI 10.1109/SANER53432.2022.00092
106. Sridhara, G.: Automatically Detecting the Up-To-Date Status of ToDo Comments in Java Programs. In: Proceedings of the 9th India Software Engineering Conference. ACM (2016). DOI 10.1145/2856636.2856638
107. Sridhara, G., Hill, E., Muppaneni, D., Pollock, L., Vijay-Shanker, K.: Towards Automatically Generating Summary Comments for Java Methods. In: Proceedings of the IEEE/ACM international conference on Automated software engineering. ACM (2010). DOI 10.1145/1858996.1859006
108. Standish, T.A.: An Essay on Software Reuse. IEEE Transactions on Software Engineering **SE-10**(5), 494–497 (1984). DOI 10.1109/tse.1984.5010272
109. Storey, M.A.D., Fracchia, F.D., Müller, H.A.: Cognitive Design Elements to Support the Construction of a Mental Model during Software Visualization. In: Proceedings 5th International Workshop on Program Comprehension. IWPC'97. IEEE Comput. Soc. Press (1999). DOI 10.1109/wpc.1997.601257
110. Sweller, J.: Cognitive Load Theory. pp. 37–76. Academic Press (2011). DOI https://doi.org/10.1016/B978-0-12-387691-1.00002-8. URL `https://www.sciencedirect.com/science/article/pii/B9780123876911000028`
111. Takang, A.A., Grubb, P.A., Macredie, R.D.: The Effects of Comments and Identifier Names on Program Comprehensibility: An Experimental Investigation. J. Program. Lang. **4**, 143–167 (1996)
112. Tan, L., Yuan, D., Krishna, G., Zhou, Y.: /*icomment. In: Proceedings of 21st ACM SIGOPS Symposium on Operating Systems Principles. ACM (2007). DOI 10.1145/1294261.1294276
113. Tan, S.H., Marinov, D., Tan, L., Leavens, G.T.: $@$tComment: Testing Javadoc Comments to Detect Comment-Code Inconsistencies. In: 2012 IEEE 5th International Conference on Software Testing, Verification and Validation. IEEE (2012). DOI 10.1109/icst.2012.106
114. Tenny, T.: Procedures and Comments Vs. The Banker's Algorithm. ACM SIGCSE Bulletin **17**(3), 44–53 (1985). DOI 10.1145/382208.382523
115. Tenny, T.: Program Readability: Procedures Versus Comments. IEEE Transactions on Software Engineering **14**(9), 1271–1279 (1988). DOI 10.1109/32.6171
116. Tiarks, R.: What Maintenance Programmers Really Do: An Observational Study. In: Workshop on Software Reengineering, pp. 36–37. Citeseer (2011)
117. Uwano, H., Nakamura, M., Monden, A., Matsumoto, K.i.: Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. In: Proceedings of

the 2006 Symposium on Eye Tracking Research and Applications. ACM Press (2006). DOI 10.1145/1117309.1117357

118. Veltman, J., Jansen, C.: The Role of Operator State Assessment in Adaptive Automation. TNO Defence, Security and Safety (2005)
119. Vermeulen, A., Ambler, S.W., Bumgardner, G., Metz, E., Misfeldt, T., Thompson, P., Shur, J.: The Elements of Java (TM) Style. 15. Cambridge University Press (2000)
120. Wallace, R., Bansal, A., Karas, Z., Tang, N., Huang, Y., Jia-Jun Li, T., McMillan, C.: Programmer Visual Attention During Context-Aware Code Summarization. IEEE Transactions on Software Engineering **51**(5), 1524—1537 (2025). DOI 10.1109/tse.2025.3554990. URL `http://dx.doi.org/10.1109/TSE.2025.3554990`
121. Wang, A.I., Arisholm, E.: The Effect of Task Order on the Maintainability of Object-Oriented Software. Information and Software Technology **51**(2), 293–305 (2009). DOI 10.1016/j.infsof.2008.03.005
122. Wiedenbeck, S., Evans, N.J.: Beacons in Program Comprehension. ACM SIGCHI Bulletin **18**(2), 56–57 (1986). DOI 10.1145/15683.1044090
123. Wilcoxon, F.: Individual Comparisons by Ranking Methods. Biometrics Bulletin **1**(6), 80–83 (1945). URL `http://www.jstor.org/stable/3001968`
124. Wong, E., Yang, J., Tan, L.: AutoComment: Mining Question and Answer Sites for Automatic Comment Generation. In: 2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE). IEEE (2013). DOI 10.1109/ase.2013.6693113
125. Woodfield, S.N., Dunsmore, H.E., Shen, V.Y.: The Effect of Modularization and Comments on Program Comprehension. In: Proceedings of the 5th International Conference on Software Engineering, ICSE '81, pp. 215–223. IEEE Press, San Diego, California, USA (1981)
126. Ying, A.T.T., Wright, J.L., Abrams, S.: Source Code That Talks: An Exploration of Eclipse Task Comments and Their Implication to Repository Mining. ACM SIGSOFT Software Engineering Notes **30**(4), 1–5 (2005). DOI 10.1145/1082983.1083152
127. Yusuf, S., Kagdi, H., Maletic, J.I.: Assessing the Comprehension of UML Class Diagrams Via Eye Tracking. In: 15th IEEE International Conference on Program Comprehension (ICPC '07). IEEE (2007). DOI 10.1109/icpc.2007.10
128. Zheng, R.Z.: Cognitive Load Measurement and Application. New York, NY **10**, 9781315296258 (2017)