Code Comprehension Beyond Best Practices: Exploring the Developer Cognitive Spectrum

Faith Culas
University of Auckland
Auckland, New Zealand

Reid Holmes
University of British Columbia
Vancouver, Canada

Thomas Fritz
University of Zurich
Zurich, Switzerland

Priyanka Dhopade
University of Auckland
Auckland, New Zealand

Kelly Blincoe
University of Auckland
Auckland, New Zealand

Abstract

Code comprehension is a cognitively demanding task, becoming increasingly important as developers spend more time reviewing AI-generated code. Prior research has examined factors influencing comprehension, including source code characteristics, developer traits, practices such as refactoring, and different strategies developers use to comprehend code. Despite this work, there is no clear consensus on what effectively supports comprehension. Refactoring standards still rely largely on static code metrics, many best practices lack empirical validation, and findings are often contradictory. More fundamentally, given the cognitive nature of code comprehension, the cognitive reasons why developers adopt different comprehension strategies remain largely unexplored. In this study, we examine how individual cognitive differences impact code comprehension by looking at how refactoring affects developers' cognitive processes. We performed a within-subjects eye-tracking experiment where student developers view four Java tasks, either a refactored or non-refactored version, allowing us to compare how code refactoring affects their cognitive behaviour. First, we use the GenderMag cognitive framework to characterize developers' cognitive styles. Then, during code comprehension tasks, we use eye-tracking to observe their cognitive behavioural patterns and analyze whether these patterns differ or align across cognitive styles. Our preliminary results with four pilot participants showed clear variations in their code-tracing strategies across different cognitive styles. Addressing this gap in understanding developer cognitive behaviour in code comprehension provides insights into the needs of developers through empirical investigation. This research calls for a fundamental shift from generic best practices toward developer-centric best practices—ones that are aware and recognize individual differences across the software engineering workflow.

Keywords

Code comprehension, Cognitive diversity, Modularity, Decomposition, Software engineering best practices, Human aspects

Conference'17, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

ACM Reference Format:
Faith Culas, Reid Holmes, Thomas Fritz, Priyanka Dhopade, and Kelly Blincoe. 2026. Code Comprehension Beyond Best Practices: Exploring the Developer Cognitive Spectrum. In . ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Code comprehension is a complex cognitive task that is estimated to consume 30–70% of developers' time [28, 33]. Its importance is increasing as developers spend more time reviewing AI-generated code than writing code themselves [34]. Code comprehension goes beyond reading lines of code; it involves constructing a mental model of how the code works and is essential for tasks such as debugging, code review, and maintenance. Prior research has identified factors that influence code comprehension, including source code characteristics (e.g., comments and identifier names) and individual developer characteristics (e.g., age and experience) [64]. Despite these insights, many widely accepted coding best practices in code comprehension still lack empirical validation, and existing findings often contradict each other, creating uncertainty about which practices genuinely improve comprehension. For example, Robert C. Martin advocates writing short functions that do one thing [27], a principle commonly applied through the *Extract Method* refactoring [16]. While this refactoring principle is believed to improve code design and comprehensibility, studies report that it can instead increase code smells [8]. Ordoñez-Pacheco et al. [37] highlight that many practices are accepted because they have been successful in specific contexts, but this does not necessarily make them universal. They conclude their study highlighting that there is still a lack of formalisation and empirical validation of what we call best practice in software engineering (SE), and emphasize the need to better understand the nature of these practices and how we evaluate them.

While the importance of code comprehension and the contradicting findings are a good enough motivation for the very active research in the area, we believe that a very important factor has often been overlooked—the underlying cognitive differences among developers. Wyrich et al. [64] in their systematic mapping study of 40 years of code comprehension experiments mention that the cognitive process of code comprehension is still fairly under-explored. Although popular code comprehension models examine comprehension strategies (top-down and bottom-up) and cognitive factors (working memory and cognitive load), they have yet to account for why individuals adopt different comprehension strategies [19]. That said, a growing body of work has explored the effect of individual

demographic differences such as age [23], gender [23], neurological conditions [30], experience [6, 14, 24, 45, 47, 53, 62, 63, 65], and code familiarity [22, 63]. While various human factors have been explored, the underlying cognitive styles of the developers that can explain these differences in depth are yet to be explored.

In addition to this gap in literature, there is a also fundamental challenge in how code comprehension is measured. Due to its subjective nature, the definition of code comprehension is often implicitly shaped by the chosen measurement, such as time taken to understand code, accuracy in answering comprehension questions, or the quality of a written summary [64]. As a result, evaluation methods for code comprehension are frequently ambiguous and lack concrete guidelines [48, 64]. Code summarisation is a good example of this issue. It is commonly used for measuring code comprehension, yet the field still lacks agreement on what constitutes a "good" code summary. Existing evaluation approaches vary widely, ranging from rule-based heuristics [56], to identifying important code elements [59], to structural analyses based on Abstract Syntax Trees (ASTs) [9, 26]. However, these methods rely on different assumptions about what matters for comprehension, such as summary length, level of abstraction, or selected code elements. Moreover, what developers perceive as important in code can differ substantially [43], suggesting that subjective judgment and individual comprehension styles may influence both code summaries and its evaluation. This lack of consensus is particularly concerning given the growing importance of code summarisation in practice as it is one of most frequently used features of AI assistants [50]. It is therefore important to understand how developers' cognitive styles and code comprehension strategies influence code summarisation, and how these factors, in turn, shape our interpretation of a good code summary.

Motivated by these gaps, we hypothesize that both code comprehension strategies and code summaries are subjective and differ based on developers' cognitive differences. With this aim, in this journal ahead workshop paper, we perform an eye-tracking experiment to understand the cognitive variability in how developers approach code comprehension and code summarisation tasks. We do so by looking at one of the popular best practices in code refactoring—**functional decomposition** in the form of *Extract Method*. For instance, a common indicator for method extraction is method length (e.g., more than 20 lines). However, it remains unclear whether this level of abstraction benefits all cognitive styles. Our study is built on the preliminary work of Tempero et al. on the comprehensibility of functional decomposition [61]. Their study on whether functional decomposition enhances code comprehensibility yielded negative results. Despite the limitations explained, yet one important factor remains underexplored—the cognitive styles of the developers. We do not argue that decomposition is not beneficial at all. While it is widely used for reasons such as reusability, readability, and maintainability, we want to investigate the relationship between code comprehension and developer cognitive styles. Our research is guided by two research questions:

RQ1: How do code comprehension strategies vary for different decompositions of functions across developer cognitive styles? We investigate how participants comprehend a version of four Java tasks, through several factors such as eye gaze data, participant self-assessments, code summarisations, and correctness of answers. During the experiment, eye tracking captures developers' code tracing styles via fixations and scanpaths, revealing their comprehension strategies. We administer a pre-experiment GenderMag questionnaire to assess their cognitive styles related to problem solving [5]. We then analyse how comprehension strategies vary across the developers' cognitive styles.

RQ2: How do developers' code comprehension based on code summaries relate to their cognitive styles and comprehension strategy? To assess code comprehension, participants are asked to summarise what the code does for each task. We then investigate whether these summaries are related to developers' cognitive styles. For example, if a developer is risk-averse, does this cognitive trait shape how they trace code when code is decomposed versus when it's not? And if code-comprehension strategies vary because of diverse cognitive styles, might this, in turn, lead to differences in code summarisation?

In summary, this paper with its extension as a journal, plans to makes the following contributions:

(1) To the best of our knowledge, we are the first to investigate how cognitive styles influence code comprehension to understand why developers vary in code comprehension strategies. This work establishes a starting point for studying the role of cognitive styles in SE tasks and underscores their importance in the design of SE tools and practices.
(2) Through empirical evidence, we present more insights into what a "good" summary means for developers. As AI generated summaries become common, this study can inform the design of AI coding assistants and best practices for code comprehension, such that different developers should have access to different types of summaries and code structure through personalisation. This also serves as a call for action on existing evaluation metrics for coding practices and code comprehension studies.
(3) We conduct a controlled experiment using realistic task conditions, including an IDE and substantial code examples, rather than isolated snippets or images common in prior work. Our eye-tracking setup even supports collapsing methods, uncommon in such studies. Comprehension is measured through realistic tasks rather than testing factors such as recall, better approximating how developers actually work with code. This exemplifies moving comprehension studies toward more naturalistic settings.

The remainder of this paper is organised as follows: Section 2 presents background and related work. Section 3 details the methodology. Section 4 presents preliminary results. Section 5 discusses these results in an exploratory manner. Section 6 addresses threats to validity. Section 7 concludes the paper.

2 Related work

This section reviews related work on code comprehension, summarization, and the cognitive aspects explored in previous studies.

2.1 Static code metrics and code comprehension

Research spanning decades has identified numerous static code characteristics that influence code comprehension, including naming conventions, lines of code, and code comments [64]. These are

often used to guide refactoring to ease comprehension. Yet contradictory findings persist regarding their relative importance. For instance, while the majority of studies support the finding that meaningful long identifiers improve comprehension, Beniamini et al. found results that single letter variables can be beneficial in certain contexts and lead to more concise code [3]. In contrast, Teasley found that naming style has no effect on comprehension and emphasizes the need to better understand the characteristics of the comprehender of the code [60]. A study by Scalabrino et al. [47] with 121 code metrics which resulted in negative results highlights that each developer has their unique process for understanding code. Interestingly, they found that some developers found the least understandable code snippet to be understandable. Individual developer cognitive characteristics remain under-explored, leaving unresolved how cognitive differences shape comprehension alongside source code properties.

2.2 Modularity and code comprehension

Guided by static code metrics, modularity is a widely practised mechanism. It involves structuring programs into smaller components such as modules, classes, or methods, and is commonly believed to enhance comprehensibility and support local changes without affecting other components [38]. Functional decomposition is a common application of modularity, whereby a task implemented as a single method is refactored into multiple smaller methods. Decomposition is often motivated by factors such as long methods, which are frequently treated as a code smell. The *Extract Method* refactoring—often described as the "Swiss army knife of refactorings"—is one of the most widely used techniques for decomposing long code fragments, reducing complexity, and eliminating duplication [2]. However, empirical evidence on the effects of decomposition on code comprehension remains mixed. Segalotto et al. [49] reported higher correctness when participants comprehended non-modularized code, while Tempero et al. [61] found no clear relationship between functional decomposition and comprehension. Ribeiro et al. [42] observed that novices reported better comprehension with larger code snippets, a preference not shared by experts. Additionally, Segalotto et al. [49] found that although developers expend less cognitive effort when understanding modularized code, they spend more time doing so, without corresponding gains in correctness. A recent large-scale study analyzed over 600k methods and found that code metrics used in prior studies exhibit high variance, particularly in small code samples explaining why prior results are often conflicting [10]. These findings suggest that factors beyond modularity may influence comprehension outcomes, leaving the relationship between decomposition and code comprehension an open research question.

2.3 Cognitive processes in code comprehension studies

Advances in neuroscience and cognitive psychology have enabled empirical studies of code comprehension. Techniques such as EEG [17, 24, 31] and fMRI [7, 40] have linked programming tasks to brain regions involved in working memory, attention, and language processing, while eye-tracking studies [6, 41, 54, 55] have revealed patterns of visual attention and reading strategies. Studies report conflicting findings on cognitive processes as well. For example, one study shows that developers focus more on the method body [1] in contrast to a prior study which found that developers primarily focus on the method signature [44]. Teasley [60] highlights that comprehension depends on *who* is doing it, and in line with this, even though prior work has looked at the characteristics of the developer (the who), there is still less knowledge on how their underlying cognitive styles influence code comprehension, leaving open questions about why developers use different strategies. Wyrich et al. [64] conclude their systematic mapping study of 40 years of code comprehension, by emphasizing that research on the cognitive processes involved in code comprehension is essential to guide study designs and explain developers' strategy choices.

2.4 Evaluation metrics in code comprehension studies

When it comes to evaluating code comprehension, studies have used diverse metrics, including correctness, time to completion, recall ability, and code-based tasks such as bug finding [64]. Among these, correctness and time are the most commonly used [36]. However, both present limitations: when summarisation is used as a measure of correctness, the quality is inherently subjective [36], while time-to-completion is ambiguous, as longer durations may reflect either task difficulty or deeper analysis rather than poor comprehension [20, 39, 61]. Among these evaluation metrics, code summarisation is particularly promising, as it captures how developers interpret in natural language, and its relevance is underscored by its widespread use in industry, including AI coding assistants [50]. Recent research in code summarisation focus on incorporating human attention data into machine learning models [57, 66]. Despite these advances, human-generated summaries continue to outperform machine-generated ones and research is exploring how to align AI models with human focus [25]. But developers do not attend to code uniformly. Their subjective judgment affects the quality of the summary [43]. This highlights that while machine aims to mimic human attention, our understanding of differences in human comprehension itself remains limited, leaving a critical gap in our understanding of what developers regard as good summary. Studying these differences can help explain why summary quality varies across developers and inform the design of models and evaluation metrics that better account for the full spectrum of cognitive styles.

3 Methodology

This section describes our study design and implementation. We have completed a pilot study with four participants. The main study will follow the same setup as the pilot, recruit additional participants, and run over three months.

3.1 Study design

Using cognitive styles as a lens, this study examines how developers comprehend decomposed versus non-decomposed code. We aim to inform refactoring best practices and, more broadly, emphasize the role of cognitive styles in SE tools and practices. We structured the experiment around four key aspects of code comprehension studies [15]:

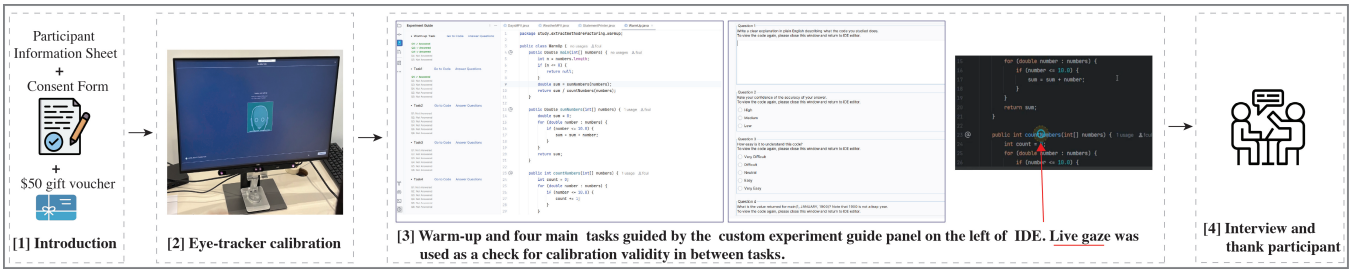

Figure 1: The experiment flow starting from introduction and calibration to the tasks and finally ending with the interview.

(a) Code selection: We limited the study to Java, widely taught in SE curricula and used in industry. Code snippets were drawn from prior work and popular refactoring exercises (see Section 3.4).

(b) Participant selection: We recruited students, representing novice or early-career developers. By studying novices, we aim to capture cognitive differences before they are partially suppressed or compensated for through experience, thereby enabling a fairer evaluation. Prior research supports the use of student participants [13, 46] but a key consideration is the population they represent, which we address by collecting data on participants' programming experience levels.

(c) Task selection: The experiment was limited to four code comprehension tasks, so that sessions remained under 90 minutes to prevent fatigue and loss of concentration.

(d) Metrics: Comprehension was assessed via summarisation and two multiple-choice questions per task, to avoid overburdening participants and thus risking loss of motivation. Eye-tracking data was collected to examine differences in comprehension strategies rather than correctness.

3.2 Participant recruitment

The four pilot study participants were recruited through immediate networks without wider advertisement. For the extended study, participants will be recruited through university online student channels and flyers targeted to computer science and software engineering students at the institution. Anyone interested in taking part in the experiment is to fill out a short questionnaire to express their interest. This questionnaire includes questions to assess participants' cognitive style, gender, and programming experience. Programming experience is collected for both general programming and Java-specific experience using years of experience and self-reported skill levels on a five-point scale ranging from novice (stage 1) to expert (stage 5) [12]. To assess cognitive styles, we use GenderMag, a well-established method for assessing cognitive styles related to problem solving [4]. GenderMag has been applied to identify biases in SE tools, such as code review tools [35], GitHub [32], and PyCharm debugging [11]. GenderMag uses three personas—Abi, Pat, and Tim—to represent five cognitive facets: Motivations, Computer Self-Efficacy, Information Processing Style, Learning Style, and Attitudes toward Risk (see Figure 2 for a description of the facets for each persona). Individuals fall along a continuum for each facet. GenderMag includes a validated likert-scale questionnaire and uses median scores to classify participants into personas [18]. For our pilot study, we used the median of a similar sample of students responses from a previous study (see Figure 3 for the personas of the four pilot participants).

Facet	Abi	Pat	Tim
Motivation	Uses technology only as needed for the task. Prefers familiar features	Exhibits both Abi and Tim characteristics	Uses technology to learn new features
Self efficacy	Lower self-efficacy than their peers. Often blames self and might give up as a result	Medium self-efficacy. Keeps trying for a while when problems arise	Higher self-efficacy than peers. Usually blames technology when problems arise. Tries multiple ways before giving up
Information processing style	Gathers and read comprehensively before taking action	Gathers and read comprehensively before taking action	Reads selectively. Follows any cues and backtracks
Learning style	Process oriented	Exhibits both Abi and Tim characteristics. Tries new features but does so mindfully	Tinkerer but this can be distracting
Risk attitude	Risk-averse	Risk-averse	Risk-taker

Figure 2: GenderMag cognitive facet description for each persona

Figure 3: Participants categorised into GenderMag personas

3.3 Study procedure

Each participant completed a 90-minute individual lab session. The session began with a brief introduction, followed by eye-tracker calibration. Participants then completed a short warm-up task to familiarise themselves with the environment and setup. This was followed by the main observation tasks (Section 3.4), during which screen activity, interactions, and verbalisations were recorded. The session concluded with a short retrospective interview. The study was approved by the institution's ethics committee.

1) Introduction: We began by introducing participants to the study goals, session structure, and experimental setup. A $50 gift voucher was given as an appreciation for their participation. Written informed consent was obtained. We also recorded whether participants wore glasses or contact lenses, as this can affect eye-tracking accuracy.

2) Calibration: We calibrated the eye tracker using an 8-point procedure [52], performed once at the start of the session. To monitor calibration quality, we developed a live gaze visualization integrated into the IDE (see Figure 1)

3) Observation: The observation phase began with a brief warm-up task (approximately five minutes) to familiarize participants with the recording setup, IDE, and task format. Participants then completed four Java tasks (Section 3.4) on IntelliJ. To provide context, we included a scenario and a deliverable:

- *Scenario:* "You are in a team of software engineers. The team has decided moving forward, it would be beneficial to add some summaries for functions in the project codebase to help new team members like yourself when they first join."
- *Deliverable:* "A summary of the method that newcomers can also use to understand what the code does and begin work faster."

For each task, participants read the code and answered six questions via an experiment guide embedded in the IDE (Figure 1). These questions included: a written explanation of the code (*"Write a clear explanation in plain English describing what the code does."*), confidence and difficulty ratings, two multiple-choice questions to motivate understanding, and a brief reflection on revisiting the code (*"If you revisited the code while responding to the questions, could you explain what made you check it again?"*). Participants could freely check the code and revise answers while working on a task, but could not return to completed tasks due to eye-tracking constraints. During the tasks, the researcher remained in the lab but moved away from the participant's immediate workspace to promote a natural working environment.

4) Interview: Finally, we conducted a semi-structured retrospective interview. Participants were shown a summary of their eye-tracking data highlighting areas of visual focus, which served as a debrief and reflection aid. Follow-up questions were used to clarify participants' code-reading strategies. A sample follow-up question would be, *"This is a visualization of your eye movements showing you spent more time on this region of the code. Can you explain why you spent time on this region?"* Refer the replication package for the full study guide [29]

3.4 Java tasks design

The four Java comprehension tasks are: **(1) Days Since**, **(2) Weather**, **(3) Theatre** and **(4) Supermarket**. Each participant will be shown either a decomposed or a non-decomposed version of each task, which we will refer to as the **multi-function version (MFV)**, the **single-function version (SFV)** respectively. They were presented to the participant in increasing order of difficulty (from Days to Supermarket).

The first two tasks are from a previous study on code decomposition by Tempero et al. [61] and involve single-class Java programs. The third task is a popular refactoring example from the first chapter of Refactoring by Fowler [16]. The fourth task is a popular kata [1] commonly used to improve coding skills. The last two tasks have multiple classes but the difference between the versions will only be in one class which contains the long method. We applied a set of consistent, semantics-preserving modifications across tasks. These included removing comments from SFV versions that could mimic decomposition, improving naming consistency, standardising method structure, and refactoring guided by SonarQube. No functionality was changed. Full details of code modifications, tasks and experimental materials are available in the replication package [29].

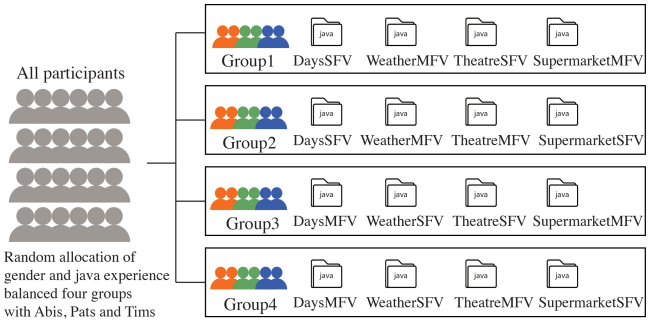

Figure 4: Participant allocation into balanced experimental groups

For task allocation, we used a within-subject factorial design with blocking, such that each participant experienced both versions (SFV and MFV). Four blocks were constructed to represent different combinations of versions. Participants were randomly assigned to one of these blocks to ensure balanced exposure to conditions and to control for potential confounding factors such as gender and experience levels (Figure 4). This design prevents participants from being exposed to multiple versions of the same code as well.

3.5 Experimental setup

The study was conducted in a closed lab space (approximately 20–30m^2), allowing the researcher to step away from the participant during tasks. Window blinds were closed to minimize sunlight, and no infrared light sources were present that could interfere with eye-tracking.

Hardware specifications: Eye-tracking data were collected using a 60 Hz Tobii Pro Spark [2], mounted on a 23.8 inch monitor, with the display mirrored for real-time monitoring. Participants and the researcher used standard mice and keyboards. Raw eye data were processed into events using the Identification by Dispersion Threshold (I-DT) algorithm, suitable for low-frequency trackers. While saccade-based metrics exist, cognitive processing primarily occurs during fixations. Also, pupil metrics are sensitive to light and often unreliable [51]; therefore, we focus on fixation-based metrics (Table 1). Screen and audio were also recorded via MS Teams, which provided automatic transcription for qualitative analysis.

Software specifications: The experiment was conducted in IntelliJ IDEA on a Windows desktop. A custom plugin managed the experimental flow within the IDE, presenting task instructions and questions to minimize researcher intervention and support natural participant behaviour (see Figure 1). Gaze data were recorded using CognitIDE [58], which maps Tobii eye-tracker data to Java source code. We extended CognitIDE to support IntelliJ build 252 and added new visualizations, including heat maps and scan paths.

[1] https://github.com/emilybache/SupermarketReceipt-Refactoring-Kata

[2] https://www.tobii.com/products/eye-trackers/screen-based/tobii-pro-spark#specifications

Table 1: Eye events and derived eye-tracking metrics

Eye Event Definition [21, 51]	Definition of the Eye metrics based on the events [21, 51]
Fixation: A stable eye gaze (typically 100–300 ms) on a specific part of a visual stimulus during which cognitive processing of that information occurs.	*Fixation Rate (%)* : Ratio of fixations in one stimulus (one task) or AOI to the total number of fixations (either in the whole stimulus or in another AOI). A lower fixation rate indicates lower search efficiency, while a higher fixation rate in comprehension tasks suggests either greater interest in the AOI or difficulty understanding it. *First fixation time* : When an AOI was first looked at.
Scan Path: The sequential pattern of fixations that traces a participant's eye movement across a visual stimulus over time.	*Attention Switching Frequency* : Number of times a participant's visual focus shifts between different AOIs, calculated as the total number of switches per minute. *Linearity* : Measures how closely eye movements follow a natural reading order (left-to-right, top-to-bottom). *ScanMatch* : A metric that compares scanpaths using the Needleman–Wunsch algorithm, adapted from DNA sequence comparison, to compute a similarity score based on fixation durations and temporal binning. *Regression Count:* A metric representing backward eye movement. Higher values indicate increased re-reading or processing difficulty.

3.6 Pilot study execution

We conducted the pilot studies in November 2025 to validate the experimental design and assess task suitability. The pilot participants included two final-year bachelor's students, one master's student, and one second-year PhD student. The latter two were included as they represent our target population of students without formal industry experience, having progressed directly from undergraduate study. Pilot sessions lasted approximately 90 minutes to two hours, longer than the planned main study sessions due to additional discussion aimed at refining the study design. Participants were informed of this in advance. Pilot data are used solely for design validation and are not included in the main analysis. Preliminary results are reported in Section 4. Based on pilot studies, we refined the experimental protocol to improve data quality and participant experience. Specifically, we increased fixation duration to 200 ms to allow adequate gaze stabilisation, simplified the calibration procedure to a single calibration per session due to its observed stability across tasks, and adjusted recording to include two recordings per task: one before answering questions and one after. All refinements are listed in our replication package [29]

3.7 Data analysis

We adopted a mixed-methods approach, combining quantitative eye-tracking metrics with qualitative responses to examine code comprehension strategies. For the pilot studies we do not provide any statistical analysis but rather present observations and comparisons to strengthen the feasibility of the study.

For a quantitative analysis of eye-tracking metrics and task answers, metrics were calculated within tasks and within predefined areas of interest (AOIs)for all four coding tasks. The full set of metrics is listed in Table 1. As method decomposition was the primary independent variable, AOIs were defined at the method level. In the SFV condition, three AOIs were defined: (1) class declaration and constants, (2) main method signature, and (3) main method body. In the MFV condition, additional AOIs were defined for each decomposed method. Descriptive statistics (mean, median, standard deviation) were computed for each metric across participants and tasks. To examine differences, we conducted exploratory comparisons of eye-tracking metrics across cognitive styles and experience levels. But for the main study this will be a statistical comparison.

For a qualitative analysis of the summaries and interviews, participants' responses and interview data were transcribed and analysed to identify patterns in code comprehension strategies. In the pilot study, one researcher conducted the analysis; in the main study, two researchers will code independently to ensure reliability. This qualitative data complements eye-tracking metrics to better understand both where participants directed their visual attention and why they used specific strategies.

4 Preliminary results

This section reports results from the four pilot sessions conducted to assess the feasibility of the study, the sensitivity of eye-tracking metrics to method decomposition, and the suitability of the proposed analysis. Given the exploratory nature and small sample size, the analyses focuses on observed trends rather than confirmatory hypothesis testing. These results are intended to inform and refine a subsequent larger-scale study. We randomly allocated the four participants to ensure coverage across all four groups. This allocation resulted in two observations per code version. For example, the DaysSFV codebase was comprehended by participants 2 and 3. In total, we collected 16 participant-code observations: eight for each condition (SFV and MFV).

4.1 Data quality and feasibility checks

Eye-tracking data were successfully collected for all four sessions. Variability was observed for both fixation-based and scanpath-based metrics, indicating sensitivity to individual differences and task structure. Figure 5 shows fixation counts and switching frequency highlighting variations across participants and tasks. Figure 6 shows fixation proportions across AOIs for two participants. Both figures reveal visible variation across participants and tasks.

4.2 Quantitative results: task performance and eye metrics

4.2.1 Task performance. Overall task performance was high. Fourteen of 16 summaries (87.5%) were correct. Multiple-choice accuracy was also high, with errors limited to Task 1: one participant answered both questions incorrectly and another answered one question incorrectly. Self-reported confidence and perceived task

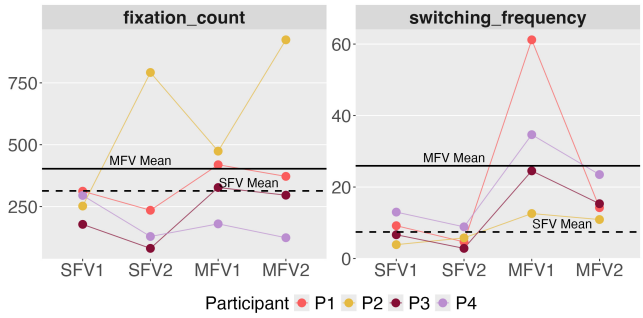

Figure 5: Fixation counts and switching frequency for participants across all tasks.

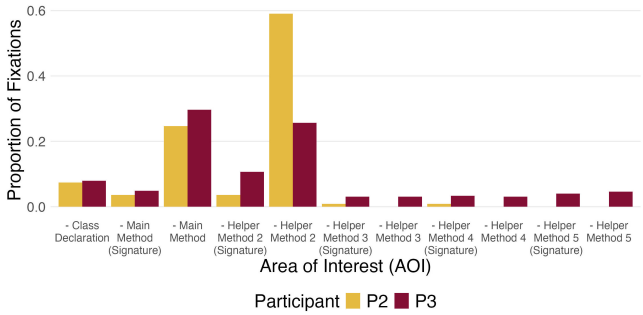

Figure 6: Fixation proportion of P2 and P3 in task2 (WeatherMFV was shown to both participants)

difficulty varied across participants. P3 expressed high confidence across all four tasks.

4.2.2 Eye metrics. Across all eye-tracking metrics, descriptive statistics revealed systematic differences between SFV and MFV tasks, with MFV tasks visibly revealing higher visual effort and more complex navigation behaviour. Also, variability within participants indicated that individual comprehension strategies differed.

Metrics related to visual effort and attention allocation, such as fixation rate and first fixation time, varied both across tasks and AOI levels. While higher fixation rates in MFV tasks suggest increased processing demands, AOI-level analysis revealed more granular differences in how participants distributed their attention. For example, P2 exhibited zero fixations on specific helper methods, indicating selective engagement rather than coverage across all AOIs (Figure 6). In terms of first fixation too we observed variations. For instance, P3 fixated on all AOIs within the first half of the task, whereas P1 initially focused only on method signatures and examined method bodies later.

As expected, attention switching frequency was substantially higher for MFV tasks (mean = 25.94) compared to SFV tasks (mean = 7.43), reflecting frequent transitions between methods rather than sustained focus on a single method. However, within-participant variability was observed, with some participants exhibiting markedly higher switching frequencies for specific MFV tasks. P1 exhibited a higher switching frequency of 60 compared to all other participants, whose values remained below 40 (see Figure 5).

Figure 7: Scanpath of P2 in task1. Color shows temporal progression starting from purple to orange to yellow.

Scanpath-based metrics revealed differences in code-tracing behaviour. Higher linearity indicated more sequential, top-to-bottom reading, while lower linearity reflected more regressions to earlier code. As shown in Figure 7, P2 exhibited highly linear code-tracing behaviour. P3 showed slightly lower linearity, with a scanpath similarity index of 0.604, indicating moderate similarity to P2 but notable differences in tracing behaviour.

Overall, these initial observations indicate that while SFV and MFV tasks differ in their fixation and scanpath metrics, participants too exhibited substantial variability in how they approached code comprehension. Differences in fixation behaviour, attention allocation, and scanpath patterns suggest that participants adopted distinct strategies even when working on the same task version.

4.3 Qualitative results: code summarisation and interviews

4.3.1 Code summarisation. Participants' code summaries ranged in length from 103 to 467 characters. We observed differences in the level of detail provided by participants with P1 being the most abstract and P4 being the most elaborative. For example, for the third task (Theatre) these were their summaries respectively: "The code provides an invoice to a performer, showing the amount of money, in USD, earned by their performance as well as the amount of volume credits earned." and "The code takes an invoice and returns the play name, a variable called thisAmount, the amount of seats purchased, the amount of money spent, and the amount of credits earned. This is returned in a formatted string. The code extracts information from the invoice and performs operations on it to get these values, with different operations being performed based on the plays genre." .

4.3.2 Interview. The interview section was valuable in validating the eye-tracking data and in providing insight into why participants traced code in particular ways. For example, P1 and P3 adopted different approaches to MFV tasks: P1 did not focus on every helper method, whereas P3 wanted to inspect each implementation. When asked about this during the interview, P1 said, *"So I think the ones that I didn't look inside, I think the I didn't really bother because like is valid day and month, that's pretty clear.... I thought well if that's what it says it does, that's what it does. So I don't have to check."* but in contrast to P1, P3 said, *"Yeah, I know. I like seeing everything. I know some people would like to still look at one at a time, but I wanted to see everything."* Triangulating eye-tracking metrics with qualitative responses revealed alignment between observed visual behaviour and reported strategies.

5 Exploratory discussion across variables

Exploratory comparisons were conducted to examine potential differences across cognitive styles and experience levels. Given the limited sample size, these observations are reported descriptively and are not interpreted as evidence of statistical significance.

Participant eye metrics and cognitive style. P2 (Abi in information processing style) emerged as a clear outlier in eye-tracking metrics, exhibiting extremely high fixation counts in SFV2 and MFV2 alongside low attention switching frequency. This combination suggests an intensive, thorough verification strategy, consistent with Abi's comprehensive information processing style. P1 and P3 (both Tim in information processing and risk attitude) showed similar trends to each other in fixation behaviour, particularly the increase from SFV to MFV tasks. While their absolute fixation values differed, their similar cognitive styles may explain similarities in code tracing and attention allocation. Notably, P1 showed an extreme peak in attention switching frequency at MFV1, indicating heightened exploratory behaviour under the MFV condition. In contrast, P4 displayed relatively higher switching frequency even in SFV tasks compared to other participants, with moderate fixation counts.

Participant eye metrics and experience levels. In relation to experience levels, P3, the most experienced participant, displayed both low fixation counts and low switching frequency, which may indicate a more goal-directed comprehension strategy with minimal visual exploration. P3 also reported high confidence across all four tasks which could have also been due to their higher level of Java experience. Overall self-reported confidence and perceived task difficulty varied across participants suggesting individual differences in how method decomposition was experienced.

Qualitative patterns and linking to metrics. Qualitative analysis reinforced the patterns observed in eye-tracking metrics. For example, P1's abstractive summaries align with moderate fixation and selective visual processing. P2's high fixation with low switching reflects a thorough strategy and aligned with longer, more elaborative summaries compared to P1.

Summary and implications. These initial observations indicate variability in code comprehension strategies that may relate to both cognitive style and experience. MFV tasks generally elicited higher fixation counts and switching frequency, but participants too exhibited distinct individual strategies consistent with some of their cognitive styles. Although conclusions cannot be derived due to the small sample, these findings demonstrate that the eye-tracking metrics are sensitive to both task structure and participants, motivating our large-scale main study to statistically evaluate these observations.

6 Threats to Validity

External validity: We recruited students, which is purposeful given our focus on novice programming behaviour. Changes in behaviour over time often reflect adaptation to industry norms rather than underlying cognitive differences, which is why we focus on novices, whose behaviour is less influenced by prior experience. However, the sample was drawn from a single institution and country, which may limit the applicability of our results to broader populations. While the study was conducted in a naturalistic setting using realistic development tools and tasks, real-world software development often involves longer time spans, collaboration, and greater complexity, which are not fully captured in our experimental setting.

Internal validity: We mitigated these threats using a controlled experimental design, random assignment, and consistent instructions and tools. The study was conducted in a controlled environment with standardised eye-tracking procedures and minimal observer interaction. Nevertheless, differences in familiarity with development environments and participant fatigue may still have influenced behaviour despite our efforts to control for these factors. Only one researcher conducted the qualitative analysis in the pilot studies, which may limit the reliability. To mitigate this, the main study will employ two independent researchers for the qualitative analysis.

Construct validity: We used eye-tracking metrics and task-related questions to capture participants' visual attention and code exploration behaviour. Although eye-tracking provides fine-grained insights into visual attention, gaze fixations do not always reflect intentional focus on a particular code element. To mitigate this, we complemented gaze data with post-task interview questions to understand participants' reasoning for fixating on specific areas. As self-reported data may be subject to recall bias, we included a question asking why participants revisited the code while answering the questions. This allowed participants to report their actions immediately, reducing reliance on retrospective recall.

7 Conclusion

This pilot study explored code comprehension strategies using eye-tracking metrics and qualitative summaries, with a focus on individual differences in cognitive style. Despite the small sample size, the analyses revealed promising trends. Notably, fixation patterns, scanpaths, and switching behaviour revealed differences in how participants allocated attention and navigated code. Qualitative summaries further reinforced these differences in comprehension strategies. Although conclusions cannot be drawn at this stage, our next step is to conduct the main study with more participants. By establishing the feasibility of the experimental setup and demonstrating the interpretability and sensitivity of the chosen metrics, this work lays the groundwork for future investigations into how individual cognitive differences shape code comprehension. Ultimately, this study can inform both empirical research methodology and the design of tools and practices that support the diverse cognitive styles of developers.

References

[1] Nahla J. Abid, Bonita Sharif, Natalia Dragan, Hend Alrasheed, and Jonathan I. Maletic. 2019. Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters. arXiv:1903.03358 [cs.SE] https://arxiv.org/abs/1903.03358

[2] Eman Abdullah AlOmar, Mohamed Wiem Mkaouer, and Ali Ouni. 2024. Behind the Intent of Extract Method Refactoring: A Systematic Literature Review. *IEEE Transactions on Software Engineering* 50, 4 (2024), 668–694. doi:10.1109/TSE.2023.3345800

[3] Gal Beniamini, Sarah Gingichashvili, Alon Klein Orbach, and Dror G. Feitelson. 2017. Meaningful Identifier Names: The Case of Single-Letter Variables. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 45–54. doi:10.1109/ICPC.2017.18

[4] M. Burnett, S. Stumpf, J. Macbeth, S. Makri, L. Beckwith, I. Kwan, A. Peters, and W. Jernigan. 2016. GenderMag: A Method for Evaluating Software's Gender Inclusiveness. *Interacting with Computers* 28, 6 (2016), 760–787. doi:10.1093/iwc/iwv046 Ed1hp Times Cited:104 Cited References Count:91.

[5] Margaret Burnett, Simone Stumpf, Jamie Macbeth, Stephann Makri, Laura Beckwith, Irwin Kwan, Anicia Peters, and William Jernigan. 2016. GenderMag: A Method for Evaluating Software's Gender Inclusiveness. *Interacting with Computers* 28, 6 (Nov. 2016), 760–787. doi:10.1093/iwc/iwv046

[6] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *2015 IEEE 23rd International Conference on Program Comprehension*. 255–265. doi:10.1109/ICPC.2015.36

[7] Joao Castelhano, Isabel C Duarte, Carlos Ferreira, Joao Duraes, Henrique Madeira, and Miguel Castelo-Branco. 2019. The role of the insula in intuitive expert bug detection in computer code: an fMRI study. *Brain Imaging Behav.* 13, 3 (June 2019), 623–637.

[8] Diego Cedrim, Alessandro Garcia, Melina Mongiovi, Rohit Gheyi, Leonardo Sousa, Rafael de Mello, Baldoino Fonseca, Márcio Ribeiro, and Alexander Chávez. 2017. Understanding the impact of refactoring on smells: a longitudinal study of 23 software projects. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn, Germany) *(ESEC/FSE 2017)*. Association for Computing Machinery, New York, NY, USA, 465–475. doi:10.1145/3106237.3106259

[9] Qiuyuan Chen, Han Hu, and Zhaoyi Liu. 2019. Code Summarization with Abstract Syntax Tree. In *International Conference on Neural Information Processing*. https://api.semanticscholar.org/CorpusID:210118029

[10] Kyle D. Chin and Reid Holmes. 2026. Put The "Code"' Back In "Code Comprehension Study"'. In *International Conference on Program Comprehension (ICPC)*. 13 pages.

[11] Faith Culas, Amisha Singh, Atharva Arankalle, Priyanka Dhopade, and Kelly Blincoe. 2026. Newcomers' experiences during debugging: A cognitive inclusivity perspective using GenderMag. *Information and Software Technology* 190 (2026), 107932. doi:10.1016/j.infsof.2025.107932

[12] Stuart E. Dreyfus. 2004. The Five-Stage Model of Adult Skill Acquisition. *Bulletin of Science, Technology & Society* 24, 3 (2004), 177–181. arXiv:https://doi.org/10.1177/0270467604264992 doi:10.1177/0270467604264992

[13] Davide Falessi, Natalia Juristo, Claes Wohlin, Burak Turhan, Jürgen Münch, Andreas Jedlitschka, and Markku Oivo. 2018. Empirical software engineering experts on the use of students and professionals in experiments. *Empirical Softw. Engg.* 23, 1 (Feb. 2018), 452–489. doi:10.1007/s10664-017-9523-3

[14] Janet Feigenspan, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2012. Measuring programming experience. In *2012 20th IEEE International Conference on Program Comprehension (ICPC)*. 73–82. doi:10.1109/ICPC.2012.6240511

[15] Dror G. Feitelson. 2021. Considerations and Pitfalls in Controlled Experiments on Code Comprehension. In *2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)*. IEEE, 106–117. doi:10.1109/icpc52881.2021.00019

[16] Martin Fowler. 2018. *Refactoring: Improving the Design of Existing Code* (2nd ed.). Addison-Wesley Professional.

[17] Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. Using psycho-physiological measures to assess task difficulty in software development. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 402–413. doi:10.1145/2568225.2568266

[18] Md Montaser Hamid, Amreeta Chatterjee, Mariam Guizani, Andrew Anderson, Fatima Moussaoui, Sarah Yang, Isaac Escobar, Anita Sarma, and Margaret Burnett. 2024. *How to Measure Diversity Actionably in Technology*. Apress, Berkeley, CA, 469–485. doi:10.1007/978-1-4842-9651-6_27

[19] Ava Heinonen, Bettina Lehtelä, Arto Hellas, and Fabian Fagerholm. 2022. Synthesizing Research on Programmers' Mental Models of Programs, Tasks and Concepts – a Systematic Literature Review. arXiv:2212.07763 [cs.SE] https://arxiv.org/abs/2212.07763

[20] Johannes Hofmeister, Janet Siegmund, and Daniel V. Holt. 2017. Shorter identifier names take longer to comprehend. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 217–227. doi:10.1109/SANER.2017.7884623

[21] Kenneth Holmqvist, Marcus Nyström, Richard Andersson, Richard Dewhurst, Halszka Jarodzka, and Joost {Van de Weijer}. 2011. *Eye tracking: A comprehensive guide to methods and measures*. Oxford University Press, United States. DS_Description: Holmqvist, K., Nyström, N., Andersson, R., Dewhurst, R., Jarodzka, H., & Van de Weijer, J. (Eds.) (2011). Eye tracking: a comprehensive guide to methods and measures, Oxford, UK: Oxford University Press..

[22] Errol R Iselin. 1988. Conditional statements, looping constructs, and program comprehension: an experimental study. *Int. J. Man. Mach. Stud.* 28, 1 (Jan. 1988), 45–66.

[23] Didem Issever, Mehmet Cem Catalbas, and Fecir Duran. 2023. Examining factors influencing cognitive load of computer programmers. *Brain Sci.* 13, 8 (July 2023).

[24] Seolhwa Lee, Danial Hooshyar, Hyesung Ji, Kichun Nam, and Heuiseok Lim. 2018. Mining biometric data to predict programmer expertise and task difficulty. *Cluster Comput.* 21, 1 (March 2018), 1097–1107.

[25] Jiliang Li, Yifan Zhang, Zachary Karas, Collin McMillan, Kevin Leach, and Yu Huang. 2024. Do Machines and Humans Focus on Similar Code? Exploring Explainability of Large Language Models in Code Summarization . In *2024 IEEE/ACM 32nd International Conference on Program Comprehension (ICPC)*. IEEE Computer Society, Los Alamitos, CA, USA, 47–51. https://doi.ieeecomputersociety.org/

[26] Chen Lin, Zhichao Ouyang, Junqing Zhuang, Jianqiang Chen, Hui Li, and Rongxin Wu. 2021. Improving Code Summarization with Block-wise Abstract Syntax Tree Splitting. arXiv:2103.07845 [cs.SE] https://arxiv.org/abs/2103.07845

[27] Robert C. Martin. 2008. *Clean Code: A Handbook of Agile Software Craftsmanship* (1 ed.). Prentice Hall PTR, USA.

[28] Yanyan Zhuang Martin K.-C. Yeh, Yu Yan and Lois Anne DeLong. 2022. Identifying program confusion using electroencephalogram measurements. *Behaviour & Information Technology* 41, 12 (2022), 2528–2545. doi:10.1080/0144929X.2021.1933182

[29] Faith Mathuranayagam Culas, Kelly Blincoe, and Priyanka Dhopade. 2026. Replication package. doi:10.17608/k6.auckland.c.8255410

[30] Ian McChesney and Raymond Bond. 2019. Eye tracking analysis of computer program comprehension in programmers with dyslexia. *Empir. Softw. Eng.* 24, 3 (June 2019), 1109–1154.

[31] J. Medeiros, R. Couceiro, J. Castelhano, M. Castelo Branco, G. Duarte, C. Duarte, J. Durães, H. Madeira, P. Carvalho, and C. Teixeira. 2019. Software code complexity assessment using EEG features. In *2019 41st Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. 1413–1416. doi:10.1109/EMBC.2019.8856283

[32] Christopher Mendez, Hema Susmita Padala, Zoe Steine-Hanson, Claudia Hilderbrand, Amber Horvath, Charles Hill, Logan Simpson, Nupoor Patil, Anita Sarma, and Margaret Burnett. 2018. Open Source barriers to entry, revisited: A tools perspective. *Proceedings of ACM ICSE conference, Sweden, May 2018 (ICSE'2018)* (2018). doi:10.475/123_4

[33] Roberto Minelli, Andrea Mocci, and Michele Lanza. 2015. I Know What You Did Last Summer - An Investigation of How Developers Spend Their Time. In *2015 IEEE 23rd International Conference on Program Comprehension*. 25–35. doi:10.1109/ICPC.2015.12

[34] Hussein Mozannar, Gagan Bansal, Adam Fourney, and Eric Horvitz. 2024. Reading Between the Lines: Modeling User Behavior and Costs in AI-Assisted Programming. In *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 142, 16 pages. doi:10.1145/3613904.3641936

[35] Emerson Murphy-Hill, Alberto Elizondo, Ambar Murillo, Marian Harbach, Bogdan Vasilescu, Delphine Carlson, and Florian Dessloch. 2024. GenderMag Improves Discoverability in the Field, Especially for Women. In *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*. ACM. doi:10.1145/3597503.3639097

[36] Delano Oliveira, Reydne Bruno, Fernanda Madeiral, and Fernando Castor. 2020. Evaluating Code Readability and Legibility: An Examination of Human-centric Studies. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 348–359. doi:10.1109/ICSME46990.2020.00041

[37] Rodrigo Ordoñez-Pacheco, Karen Cortes-Verdin, and Jorge Octavio Ocharán-Hernández. 2021. Best practices for software development: A systematic literature review. In *Advances in Intelligent Systems and Computing*. Springer International Publishing, Cham, 38–55.

[38] D. L. Parnas. 1972. On the criteria to be used in decomposing systems into modules. *Commun. ACM* 15, 12 (dec 1972), 1053–1058. doi:10.1145/361598.361623

[39] Norman Peitek, Annabelle Bergum, Maurice Rekrut, Jonas Mucke, Matthias Nadig, Chris Parnin, Janet Siegmund, and Sven Apel. 2022. Correlates of programmer efficacy and their link to experience: a combined EEG and eye-tracking study. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Singapore, Singapore) *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 120–131. doi:10.1145/3540250.3549084

[40] Norman Peitek, Janet Siegmund, Sven Apel, Christian Kästner, Chris Parnin, Anja Bethmann, Thomas Leich, Gunter Saake, and André Brechmann. 2020. A

Look into Programmers' Heads. *IEEE Transactions on Software Engineering* 46, 4 (2020), 442–462. doi:10.1109/TSE.2018.2863303

[41] Norman Peitek, Janet Siegmund, Chris Parnin, Sven Apel, Johannes C. Hofmeister, and André Brechmann. 2018. Simultaneous measurement of program comprehension with fMRI and eye tracking: a case study. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Oulu, Finland) *(ESEM '18)*. Association for Computing Machinery, New York, NY, USA, Article 24, 10 pages. doi:10.1145/3239235.3240495

[42] Talita Vieira Ribeiro and Guilherme Horta Travassos. 2018. Attributes influencing the reading and comprehension of source code – discussing contradictory evidence. *CLEI Electron. J.* 21, 1 (April 2018), 5.

[43] Paige Rodeghero, Collin McMillan, Paul W. McBurney, Nigel Bosch, and Sidney D'Mello. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th International Conference on Software Engineering* (Hyderabad, India) *(ICSE 2014)*. Association for Computing Machinery, New York, NY, USA, 390–401. doi:10.1145/2568225.2568247

[44] Paige Rodeghero, Collin Mcmillan, Paul W. Mcburney, Nigel Bosch, and Sidney D'Mello. 2014. Improving automated source code summarization via an eye-tracking study of programmers. In *Proceedings of the 36th International Conference on Software Engineering*. ACM. doi:10.1145/2568225.2568247

[45] Jonathan A Saddler, Cole S Peterson, Patrick Peachock, and Bonita Sharif. 2019. Reading behavior and comprehension of C++ source code - A classroom study. In *Augmented Cognition*. Springer International Publishing, Cham, 597–616.

[46] Iflaah Salman, Ayse Tosun Misirli, and Natalia Juristo. 2015. Are Students Representatives of Professionals in Software Engineering Experiments?. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*. IEEE, 666–676. doi:10.1109/icse.2015.82

[47] Simone Scalabrino, Gabriele Bavota, Christopher Vendome, Mario Linares-Vásquez, Denys Poshyvanyk, and Rocco Oliveto. 2021. Automatically Assessing Code Understandability. *IEEE Transactions on Software Engineering* 47, 3 (2021), 595–613. doi:10.1109/TSE.2019.2901468

[48] Ivonne Schröter, Jacob Krüger, Janet Siegmund, and Thomas Leich. 2017. Comprehending Studies on Program Comprehension. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*. 308–311. doi:10.1109/ICPC.2017.9

[49] Matheus Segalotto, Willian Bolzan, and Kleinner Farias. 2023. Effects of Modularization on Developers' Cognitive Effort in Code Comprehension Tasks: A Controlled Experiment. In *Proceedings of the XXXVII Brazilian Symposium on Software Engineering* (Campo Grande, Brazil) *(SBES '23)*. Association for Computing Machinery, New York, NY, USA, 206–215. doi:10.1145/3613372.3613387

[50] Agnia Sergeyuk, Olga Lvova, Sergey Titov, Anastasiia Serova, Farid Bagirov, and Timofey Bryksin. 2024. Assessing Consensus of Developers' Views on Code Readability. arXiv:2407.03790 [cs.SE] https://arxiv.org/abs/2407.03790

[51] Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gaël Guéhéneuc. 2015. Eye-Tracking Metrics in Software Engineering. In *2015 Asia-Pacific Software Engineering Conference (APSEC)*. 96–103. doi:10.1109/APSEC.2015.53

[52] Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, Martha Crosby, Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering 2020 25:5* 25, 5 (2020). doi:10.1007/s10664-020-09829-4

[53] Bonita Sharif, Michael Falcone, and Jonathan I Maletic. 2012. 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* (Santa Barbara California). ACM, New York, NY, USA.

[54] Janet Siegmund, Norman Peitek, Chris Parnin, Sven Apel, Johannes Hofmeister, Christian Kästner, Andrew Begel, Anja Bethmann, and André Brechmann. 2017. Measuring neural efficiency of program comprehension. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering* (Paderborn Germany). ACM, New York, NY, USA.

[55] José Aldo Silva Da Costa and Rohit Gheyi. 2023. Evaluating the Code Comprehension of Novices with Eye Tracking. In *Proceedings of the XXII Brazilian Symposium on Software Quality* (Brasília, Brazil) *(SBQS '23)*. Association for Computing Machinery, New York, NY, USA, 332–341. doi:10.1145/3629479.3629490

[56] Giriprasad Sridhara, Emily Hill, Divya Muppaneni, Lori Pollock, and K. Vijay-Shanker. 2010. Towards automatically generating summary comments for Java methods. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering* (Antwerp, Belgium) *(ASE '10)*. Association for Computing Machinery, New York, NY, USA, 43–52. https://doi.org/10.1145/1858996.1859006

[57] Sean Stapleton, Yashmeet Gambhir, Alexander Leclair, Zachary Eberhart, Westley Weimer, Kevin Leach, and Yu Huang. 2020. A Human Study of Comprehension and Code Summarization. In *Proceedings of the 28th International Conference on Program Comprehension*. ACM, 2–13. doi:10.1145/3387904.3389258

[58] Fabian Stolp, Malte Stellmacher, and Bert Arnrich. 2024. CognitIDE: An IDE Plugin for Mapping Physiological Measurements to Source Code. In *Companion Proceedings of the 32nd ACM International Conference on the Foundations of Software Engineering*. ACM, 592–596. doi:10.1145/3663529.3663805

[59] Weisong Sun, Chunrong Fang, Yuchen Chen, Quanjun Zhang, Guanhong Tao, Tingxu Han, Yifei Ge, Yudu You, and Bin Luo. 2023. An Extractive-and-Abstractive Framework for Source Code Summarization. arXiv:2206.07245 [cs.SE] https://arxiv.org/abs/2206.07245

[60] Barbee E Teasley. 1994. The effects of naming style and expertise on program comprehension. *Int. J. Hum. Comput. Stud.* 40, 5 (May 1994), 757–770.

[61] Ewan Tempero, Paul Denny, James Finnie-Ansley, Andrew Luxton-Reilly, Diana Kirk, Juho Leinonen, Asma Shakil, Robert Sheehan, James Tizard, Yu-Cheng Tu, and Burkhard Wünsche. 2024. On the comprehensibility of functional decomposition: An empirical study. *2024 Ieee/Acm 32nd International Conference on Program Comprehension(ICPC 2024)* (2024). doi:10.1145/3643916.3644432

[62] Rachel Turner, Michael Falcone, Bonita Sharif, and Alina Lazar. 2014. An eye-tracking study assessing the comprehension of c++ and Python source code. In *Proceedings of the Symposium on Eye Tracking Research and Applications* (Safety Harbor Florida). ACM, New York, NY, USA.

[63] Susan Wiedenbeck. 1991. The initial stage of program comprehension. *Int. J. Man. Mach. Stud.* 35, 4 (Oct. 1991), 517–540.

[64] Marvin Wyrich, Justus Bogner, and Stefan Wagner. 2024. 40 Years of Designing Code Comprehension Experiments: A Systematic Mapping Study. *Comput. Surveys* 56, 4 (2024), 1–42. doi:10.1145/3626522

[65] Chak Shun Yu, Christoph Treude, and Maurício Aniche. 2019. Comprehending Test Code: An Empirical Study. In *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 501–512. doi:10.1109/ICSME.2019.00084

[66] Yifan Zhang, Jiliang Li, Zachary Karas, Aakash Bansal, Toby Jia-Jun Li, Collin Mcmillan, Kevin Leach, and Yu Huang. 2024. EyeTrans: Merging Human and Machine Attention for Neural Code Summarization. *Proceedings of the ACM on Software Engineering* 1, FSE (2024), 115–136. doi:10.1145/3643732