UNIVERSIDADE FEDERAL DE PERNAMBUCO
CENTRO DE INFORMÁTICA

Nathalia Paiva Lima

An Eye Tracking Perspective on Harmfulness of Code Smells: A Systematic Literature Review and Experiment Design

Recife

2025

Nathalia Paiva Lima

An Eye Tracking Perspective on Harmfulness of Code Smells: A Systematic Literature Review and Experiment Design

Recife

2025

Ficha de identificação da obra elaborada pelo autor,
através do programa de geração automática do SIB/UFPE

Lima, Nathalia Paiva.
An Eye Tracking Perspective on Harmfulness of Code Smells: A Systematic Literature Review and Experiment Design / Nathalia Paiva Lima. - Recife, 2025.
60 p : il., tab.

Orientador(a): Leopoldo Motta Teixeira
Trabalho de Conclusão de Curso (Graduação) - Universidade Federal de Pernambuco, Centro de Informática, Engenharia da Computação - Bacharelado, 2025.
Inclui referências, apêndices, anexos.

1. software engineering. 2. code smell. 3. eye-tracking. I. Teixeira, Leopoldo Motta. (Orientação). II. Título.

000 CDD (22.ed.)

NATHALIA PAIVA LIMA

AN EYE TRACKING PERSPECTIVE ON HARMFULNESS OF CODE SMELLS: A Systematic Literature Review and Experiment Design

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia da Computação da Universidade Federal de Pernambuco, como requisito parcial para obtenção do título de bacharel em Engenharia da Computação.

Aprovado em: 10/04/2025

BANCA EXAMINADORA

_____

Prof. Dr. Leopoldo Motta Teixeira (Orientador)

Universidade Federal de Pernambuco

_____

Prof. Dr. Kiev Santos da Gama (Examinador Interno)

Universidade Federal de Pernambuco

Dedico este trabalho à minha mãe Andiara e ao meu pai Paulo, por todo amor e apoio incondicional. Ao Felipe, meu parceiro, por estar ao meu lado nos momentos mais difíceis e me dar forças para continuar. À Adora e ao Teemo, que, com presença e afeto, transformaram o cansaço em aconchego e os dias difíceis em abrigo. Aos amigos que fiz nos cursos de Sistemas de Informação e, depois da transferência, em Engenharia da Computação, que tornaram essa jornada mais rica e divertida. Ao professor Fernando Castor, meu orientador de Iniciação Científica, por me inspirar desde cedo. E ao professor Leopoldo Texeira, meu orientador de TG, por toda a orientação e confiança.

AGRADECIMENTOS

Agradeço pelo apoio financeiro do CNPq durante minha Iniciação Científica e da PROGRAD durante minha atuação como monitora na disciplina de Introdução à Programação. Essas experiências foram essenciais para minha formação acadêmica e pessoal.

ABSTRACT

This work aims to update a prior systematic review focusing on the application of eye-tracking methodologies in software engineering, particularly concerning code smells. Code smells are features within source code that are difficult to read, understand, and maintain, yet their direct harmfulness remains insufficiently explored. Through a comprehensive literature review, 28 papers were identified showcasing recent advancements in eye-tracking studies from 2015 onward, highlighting new insights and limitations. It was observed that eye-trackers are used to study model comprehension, code comprehension, debugging, and traceability tasks. Based on these insights, we propose an experimental design employing eye-tracking methodology to analyze the harmful effects of code smells.

Keywords: software engineering, code smell, eye-tracking.

SUMÁRIO

1 INTRODUCTION

Over recent years, eye-tracking technology has emerged as a valuable tool in software engineering research [1]. It allows researchers to capture and analyze the visual attention of developers while they look at different parts of the code, which can provide insights into the cognitive process of our brains during code development.

According to Martin Fowler, "code smell is a surface indication that usually corresponds to a deeper problem in the system."(M. Fowler. Refactoring: Improving the design of existing code, 2018) [2]. These smells are indicators of potential issues in the code that may not necessarily be bugs, but can lead to problems in the future, such as challenges for maintainability and increased complexity. They signal the need for further investigation in order to proactively prevent these issues.

2 METHODS

2.1 RESEARCH QUESTIONS

The following research questions were based on the study executed by Sharafi et al. [3]. The current study updates these questions and further examines their validity on the context of determining the harmfulness of code smells.

- RQ1. How many studies have been published using eye-trackers in software engineering research since the previous SLR?
- RQ2. What research topics have been explored in recent eye-tracking studies within software engineering?
- RQ3: How have recent eye-tracking studies contributed to advancements in software engineering?
- RQ4: In what ways have researchers used eye-trackers to collect and visualize quantitative measurements in recent studies?
- RQ5: What are the key limitations identified in recent eye-tracking studies within software engineering?
- RQ6: What eye-trackers are most frequently used in recent eye-tracking studies?
- RQ7: How could eye-tracking methodology be used to analyze the harmfulness of code smells?

To address them, a systematic literature review of recent studies on the usage of eye-tracking in software engineering will be conducted. This methodical approach will enable us to critically analyze existing research to identify gaps, contradictions, and potential areas for further exploration.

2.2 LITERATURE SEARCH

With Sharafi et al. [3] Systematic Literature Review (SLR) as a starting point for contributions on the usage of eye-tracking on software engineering studies, this work replicates

the search process. However, instead of using Engineering Village, an aggregation platform that integrates with multiple database portals, the search was performed directly on the ACM Digital Library and the IEEE Xplore Digital Library. These databases were chosen due to their comprehensive collections of articles in the fields of computer science and engineering and also because they offer a free alternative to Engineering Village. The search was performed on August of 2023.

2.2.1 Search Query

The search query was based on the query coined by Sharafi et al. [3], with the goal of maintaining consistency and ensuring comparability of results. Such search query aims to identify a comprehensive set of relevant studies using various terms related to eye-tracking technology and its application in software engineering. The final query used was as follows:

Código Fonte 2.1 – Search Query

```
1 ("eye-track*" OR "eye track" OR "RFV" OR "Restricted Focus Viewer")
  AND ("source code" OR program* OR UML OR model* OR representation*)
3 AND (comprehen* OR understand* OR debug* OR navigat* OR read* OR scan*)
```

Due to IEEE Xplore's limitation of allowing a maximum of 9 wildcards in a single query, the original query was split into two parts to ensure comprehensive coverage. The results of both queries were combined to ensure that there were no missing results from the query variations.

To avoid duplicating the work of the SLR previously executed by Sharafi et al. [3], only studies published after 2015 were included.

2.3 SELECTION

To ensure that only relevant studies were captured, records were selected through a systematic filtering process. Relevant studies are defined as primary studies using eye-tracking technology to study and investigate software engineering activities.

Initially, only the title and abstracts of the records were screened to remove studies that meet the following exclusion criteria:

- Not published in English.
- Papers in grey literature (i.e. not peer-reviewed).

- Do not utilize an eye-tracker.
- Unrelated to software engineering.

Following this, full-text articles were reviewed to refine the selection. Therefore, studies were removed that:

- Were not a primary study (i.e. studies that do not collect and analyze new data)
- Do not primarily use eye-tracking as research approach
- Do not investigate software engineering activities (i.e. the purpose of the study should be to research or analyze behavior, performance, or cognitive aspects.)

2.4 SNOWBALLING

For each reviewed paper, references within it were systematically reviewed, beginning with the titles and publication venues (conference proceedings or journal names) and, where necessary, a full-text analysis was conducted to ascertain relevance.

2.5 DATA COLLECTION AND ANALYSIS

Given the diversity of the included studies in terms of approaches, a narrative synthesis was conducted to summarize and analyze the findings.

Through the search on the digital libraries, the following information was obtained: authors, publication year, publisher, publication title, DOI, abstract, keywords, and author affiliations.

After reading and analyzing each paper, the following data and key insights were obtained:

- Programming language used in the study
- Number and types of participants (students, faculty members, and/or professionals)
- Eye-trackers utilized in the research
- Objective: the goal of the paper
- Methodology: how the experiment was conducted
- Results: what were the key findings

- Limitations: what limited or added challenges to the study
- Variables: which metrics were used to track the experiment

If any piece of information was missing from a paper, it was noted as "not mentioned".

After analyzing the refined selection, there was a noticeable lack of standard when it comes to describing the eye-tracking metrics. Thus, in this research, the nomenclature adopted was the one used by Bryn Farnsworth in Eye Tracking: The Complete Pocket Guide [4], that defines key eye-tracking metrics as:

- Gaze points: The raw data points that indicate a snapshot of where the user is looking on the visual stimulus at any given moment.
- Fixations: A series of gaze points within a close time range.
- Saccades: Movements between different fixations, when the eyes move and pause across different sequences.
- Areas of Interest: Specific regions of a displayed stimulus predefined by researchers.
- Perceptual span: number of characters we can recognize on each fixation, between each saccade.
- Fixation Duration: how long the user spent looking at a specific area of interest.
- Fixation sequences: The order and pattern of fixations over time.
- Time to First Fixation: The time it took for the user to first look at a specific area of interest from the experiment for the first time.
- Regression: The number of times the gaze returns to a previously viewed area, indicating re-reading or reevaluation behavior.
- Heat Maps: Visual representations that uses color coding to show the frequency and duration of gaze points on different areas.
- Pupil Size: Measurements of the dilation or constriction of the pupils.
- Blinks: The delay and frequency of blinks.

The extracted information was used to identify a set of categories, then such components were aggregated to highlight the similarities, differences, and patterns across the studies. Based on the topic domains proposed by Sharafi et al. [3], the studies were classified accordingly to the following tasks performed by participants:

- Code Comprehension: How programmers read and understand source code.
- Model Comprehension: How developers interpret models and diagrams.
- Debugging: How developers identify, locate, and fix bugs in the software.
- Traceability: How developers create and follow links between related artifacts — like requirements, designs, and code — to maintain and improve software systems.
- Collaborative interactions: How developers coordinate, share attention, and interact during collaborative work.

2.6 STUDY QUALITY AND RISK OF BIAS

One potential threat to the validity of this study lies in the absence of an additional checker. That could introduce bias or oversight, impacting the overall reliability of the results.

During the selection process of this study, in order to avoid manual mistakes, a new tool called "Review Wise"(Fig. 1) was developed to help analyze papers in an interactive way. This tool allows for a one-by-one interactive screening of the search query results, allowing papers to be flagged with a custom list of exclusion criteria. The user is then presented with a curated list of papers that do not match any exclusion criteria, and also includes a list of removed papers alongside their exclusion reasons. Review Wise is an open source software and is available at https://github.com/naftalima/review-wise.

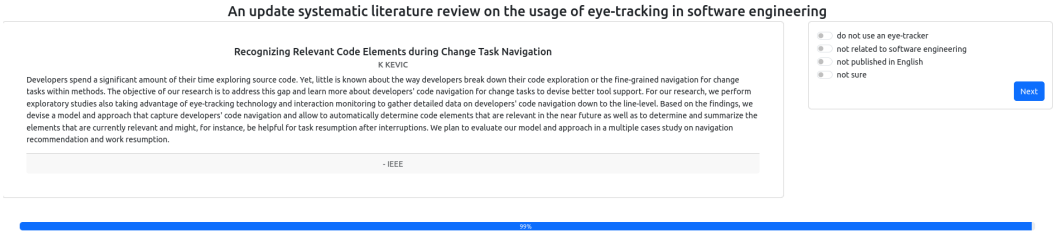

Figura 1 – Screenshot of Review Wise: a user interface showing the classification tool for applying the exclusion criteria on papers.

During the data extraction process, the relatively small number of selected studies and the objective nature of the extracted data helped to reduce errors.

Additionally, the review process was executed on two separate occasions, allowing an interval between revisions. The goal behind this was to assert a more objective evaluation of the selected studies.

3 DISCUSSION

3.1 RQ1. HOW MANY STUDIES HAVE BEEN PUBLISHED USING EYE-TRACKERS IN SOFTWARE ENGINEERING RESEARCH SINCE THE PREVIOUS SLR?

A systematic literature review was conducted using digital libraries to identify recent studies employing eye-trackers in software engineering research. The initial search returned 134 matches from the ACM Digital Library and 215 matches from the IEEE Xplore Digital Library.

To gather novel insights, building upon the previous systematic literature review by Sharafi et al. [3], the search was refined to include only studies published after 2015, resulting in the exclusion of 78 records from ACM and 42 records from IEEE.

Subsequently, a data wrangling process was conducted to ensure consistency in naming conventions and formatting, thereby facilitating the identification and removal of duplicate entries. Seven duplicate records were identified and eliminated, resulting in a total of 222 unique records. The entire data wrangling process is fully replicable and available at https://github.com/naftalima/tg

During the preliminary screening, 162 records were excluded. Approximately 8% of these records were removed because they did not utilize eye-trackers, and approximately 67% pertained to unrelated fields such as mathematics, geoinformatics, bioinformatics, and robotics. No records were excluded due to grey literature or language constraints, likely because the search was limited to well-focused databases.

Finally, the 60 remaining records underwent an in-depth, full-text analysis to further refine the selection. The remaining papers were discussed in more detail in Appendix .2. This process led to the exclusion of 35 additional records for the following reasons:

- Five papers did not collect new empirical data; instead, they reviewed existing literature, discussed future research directions, or revisited earlier studies. [5] [6] [7] [8] [9]
- One study did not primarily use eye-tracking as a research method, employed eye-tracking only as a supplement to fNIRS to investigate the impact of source code readability on cognitive load [10].
- Twenty-nine papers focused on tools, frameworks, or modeling visual attention rather than directly addressing software engineering activities. [11] [12] [13] [14] [15] [16] [17]

[18] [19] [20] [21] [22] [23] [24] [25] [26] [27] [28] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39]

A snowballing approach was applied and 3 additional studies were identified [40] [41] [42].

Consequently, a total of 28 relevant studies using eye-trackers in software engineering research have been identified since the previous SLR. Figure 2 summarizes the Systematic Literature Review process performed.

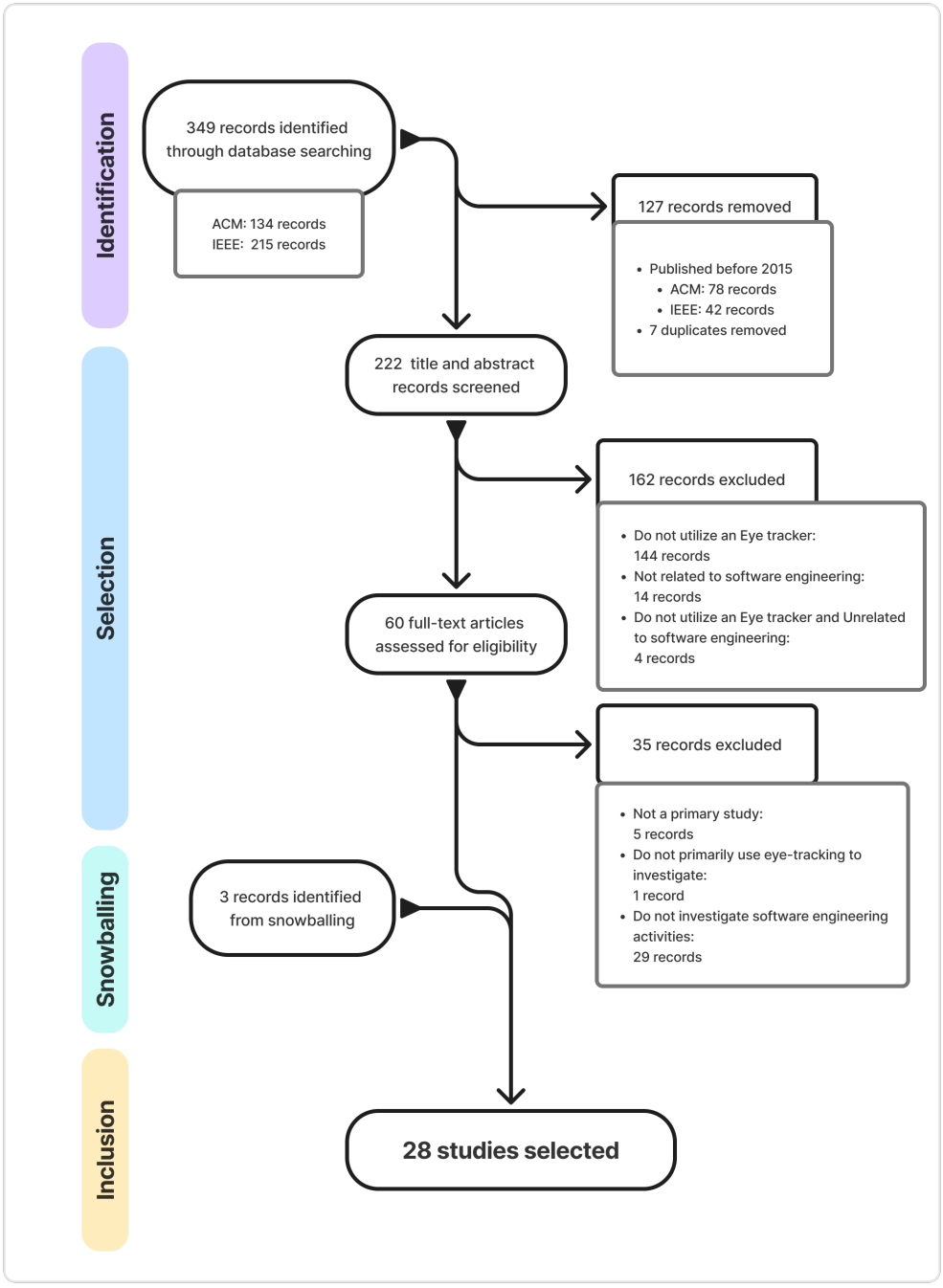

Figura 2 – Systematic Literature Review workflow

3.2 RQ2. WHAT RESEARCH TOPICS HAVE BEEN EXPLORED IN RECENT EYE-TRACKING STUDIES WITHIN SOFTWARE ENGINEERING?

resented on the methods. Since there were no papers identified as Collaborative Interactions, the remaining selection was categorized into four distinct groups: Code Comprehension, Debugging, Model Comprehension, and Traceability.

As shown in Fig. 3, code comprehension has been the most consistently studied topic domain throughout the years, with a notable peak in 2019. However, there is a visible decline in overall publications from 2020 onward (Fig. 4), which may be partially attributed to the impact of the COVID-19 pandemic on human subjects research [71], such as eye-tracking research.

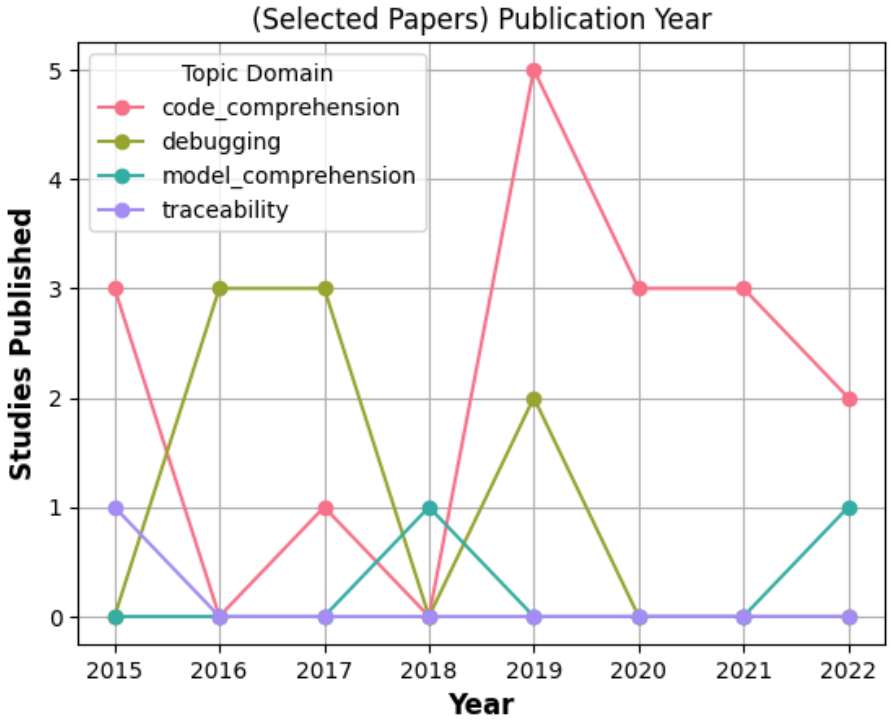

Figura 3 – Number of studies published per topic domain per year, based on selected papers from the SLR.

3.2.1 Code Comprehension

Code comprehension studies explore how programmers read and interpret source code. Within this research domain, a total of 17 studies analyze navigation strategies, emphasizing

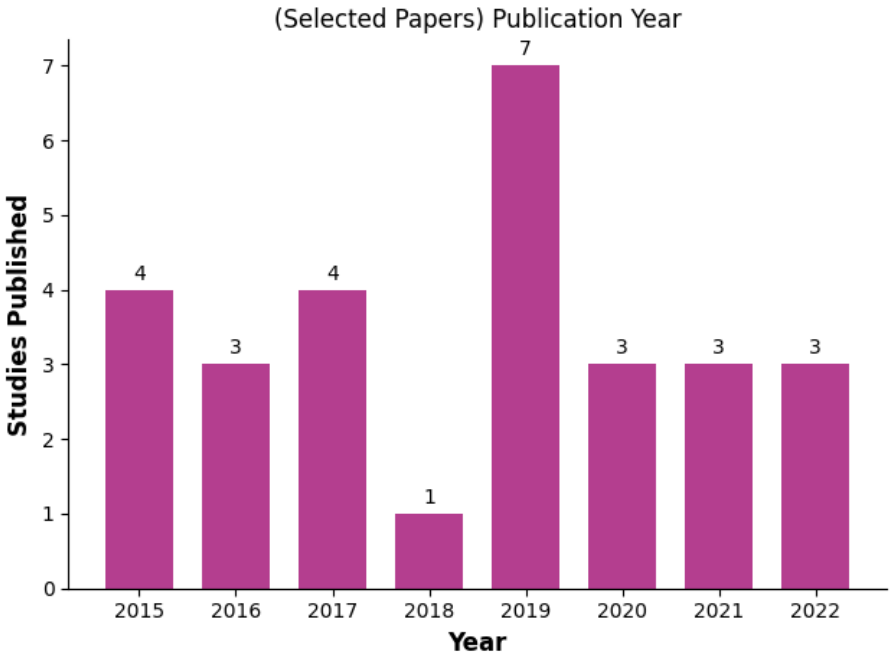

Figura 4 – Number of studies published per year, based on selected papers from the SLR.

specific aspects of code or programmer backgrounds.

Researchers have investigated how code structure, formatting, and syntax influence readability. Studies on code structure evaluate how code organization affects readability, considering factors such as repetitive code [43], nesting and the use of do-while loops [44], and source code linearity—particularly its impact on readability and reading order [45].

Code formatting pertains to the visual presentation and style, including elements such as indentation levels [46] and comparing crowded versus spaced layouts [47]. Code syntax analyses examine specific syntactic elements like LINQ's query-based versus method-based syntax [48], the use of different language constructs [49], and the effects of polyglot programming (using multiple languages within a single project) versus monoglot programming (using a single language) [50].

Additionally, studies have considered how programmers learn new language concepts, including specialized executable specification languages such as CASM [51].

Moreover, researchers investigate the reading process by comparing how programmers comprehend source code versus natural language text [40] [52] and how they interact with code during practical tasks. Such tasks include: leveraging Stack Overflow discussions for code comprehension and summarization [42] [53]; examining how correlated information is better display in supporting documentation, such as the example captured in [54] that explores how embedding security information within the API documentation influences developers' unders-

tanding and their ability to write secure code. Furthermore, researchers extend their analysis by considering developers themselves by investigating the influence of programming expertise when comparing navigation strategies employed by novices and experts [45] [40] [52] [55], as well as dyslexia on code comprehension [47].

3.2.2 Debugging

These seven debugging studies explore how programmers identify and resolve software bugs by examining the influence of both the nature of issues and developer experience on debugging strategies.

Research in this area includes studies analyzing how programmers navigate through code to locate errors, investigating the effectiveness of error messages in facilitating debugging [56], evaluating different debugging approaches depending on the type of bug [57], and examining challenges when debugging software with variability introduced by conditional compilation flags [58] [59]. Further studies consider the role of developer expertise by comparing the debugging strategies employed by novices and experts [60], as well as between high and low performing programmers [61].

3.2.3 Model Comprehension

Two studies on model comprehension explore how developers interpret models and diagrams. Andrzejewska and Stolińska [62] evaluate efficiency when solving algorithmic tasks presented as pseudocode versus flowcharts, and Störrle [63] investigating how modelers read and interpret UML diagrams.

3.2.4 Traceability

Research on traceability investigates how developers track or maintainlinks between related artifacts—such as requirements, designs, and code— over time or across systems. Kevic [41] examines developers' detailed behavior while performing software change tasks.

3.3 RQ3: HOW HAVE RECENT EYE-TRACKING STUDIES CONTRIBUTED TO ADVANCEMENTS IN SOFTWARE ENGINEERING?

3.3.1 Source code vs. natural text

Programmers read source code less linearly than natural text

Peachock et al. [40] replicated Busjahn et al. [52] original study, and both found that programmers read source code significantly less linearly than natural language text, with more regressions. However, they diverged on whether programming expertise influences reading behavior.

Clear documentation with code examples enhances comprehension

Saddler et al. [53] show how developers leverage Stack Overflow discussions. This study found that clear text combined with code examples draws the most attention, while longer paragraphs reduce focus on comments. It also found that developers who are used to use Stack Overflow spend less time on querying for possible solutions and navigate through the website faster.

Peterson et al. [42] describes how developers primarily focus on the body text and code snippets of Stack Overflow posts, and they rarely view titles, tags, or votes, highlighting that text and code are most valuable for tasks like code summarization.

Effective documentation includes relevant examples

In Gorski et al [54], developers were found to mainly focus on code examples and skim the rest of the documentation. In this experiment, the API specification was presented alongside information on how to make access to such API in a secure manner. What was discovered was that placement matters: security info near functional examples is more effective in promoting secure integration for applications onboarding to this API. As a guidance, API providers should embed security guidance near core examples to support secure coding.

Code reviews require attention to comments and error lines

Chandrika et al [66] compares the visual attention of subjects with programming skills and subjects without programming skills, and recognizes the eye-tracking traits required for source code review. During this, it was found that the attention span on error lines and comments, as well as better code coverage, was a key aspect for code review.

3.3.2 Navigation strategies

Developers do not read the code linearly from top to bottom, but explored lines are connected through code flow

Peterson et al. [49] highlight that developers focus their attention on certain small sections of the code instead of scanning the entire program. Kevic et al. [41] observed that while performing a change task, developers focus on small portions of methods, and explored lines are typically connected through data flow.

Rodeghero et al. [64] and Abid et al. [65] agree that developers do not read code linearly from top to bottom, but rather selectively allocate their attention to specific code elements or areas.

Rodeghero et al. [64] finds that programmers read method signatures more closely than method bodies or invocations and states that developers pay the least attention to control flow statements. Abid et al. [65] argues the opposite, suggesting programmers pay more attention to the method body rather than the signature, and instead observed programmers frequently revisit control flow terms, indicating more repeated interactions with control flow structures.

The effort of reading error messages is comparable to reading source code

Barik et al. [56] provides empirical justification for the need to improve compiler error messages. The developers are reading error messages, but spent a substantial amount of time understanding these error messages, despite the fact that most of them had only a single error present.

There is a more effective debugging strategy for each type of bug

Peng et al. [57], found that for data flow bugs, it was more helpful to dedicate attention to how the data values change during debugging. As for control flow bugs, the better approach was to dedicate attention at the source code to understand the logical structure.

Katona et al. [67], found different debugging approaches were observed: those who attempted to find errors with many minor changes, and those who first interpreted the source code.

Lin et al. [61], found that low-performance students tend to debug programs aimlessly, while high-performance students debug programs in a more logical manner.

Structured flowcharts improve task-solving efficiency and accuracy

Andrzejewska and Stolińska [62] found solving tasks with a structured flowchart required less time for re-examining the algorithm and data, leading to more accurate outcomes than when using pseudocode. The study indicate that the use of structured flowcharts in various educational materials, including textbooks, should be encouraged.

Diagram complexity and layout flaws increase cognitive load

Störrle et al. [63] found that ordinary diagram elements and layout flaws increased cognitive load. Also, no patterns were identified for activity and class diagrams with poor layout, use case diagrams regardless of layout quality, or expertise levels.

3.3.3 Developer Background

Experience modulates the linearity of reading

According to Beelders [55], novices read code more linearly, focusing on method signatures and comments, whereas experts efficiently skip irrelevant parts, focusing more on complex logic. Peachock et al. [40] and Busjahn et al.'s [52] diverged on that, but there were some differences in their experiment design.

In replication, Peachock et al. [40] used code snippets with different programming languages, and a different definition of expertise, that did not invite expert programmers, but advanced undergraduate students, which they refer to as "non-novice" participants. As a result, no significant differences in reading behavior between novices and non-novices were found.

Nivala et al. [60] found that on error finding tasks, novice developers spent more time reading the code, while experts identified issues sooner and spent more time describing the error.

The type of language variant, polyglot or monoglot programming, can increase understanding depending on the type of task

In Peterson [50], the type of language variant (monoglot, embedded SQL, or hybrid) did not significantly affect how long participants took to complete the tasks. Developer gaze behavior — used as a proxy for cognitive effort and processing — was influenced by the language context, particularly depending on the type of task.

Prior concept knowledge outweighs experience in understanding new programming languages

According to Simhandl et al. [51], foreknowledge of specific programming concepts rather than the programming experience is decisive in learning new programming language concepts.

Dyslexia does not reduce programming ability

McChesney and Bond [47] investigate how code layout (crowded vs. spaced) impact code comprehension, particularly in programmers with dyslexia. The experiment found no significant evidence that programmers with dyslexia are more negatively affected by code crowding compared to non-dyslexic programmers. Therefore, dyslexia is not a detrimental condition when it comes to programming, at least in terms of code reading and program comprehension.

3.3.4 Code layouts, complex code structure, and syntax

Linearity of source code modulates reading behavior

Peitek et al. [45] decided to investigate the influence of the linearity of the source code itself. It also found that experience modulates the reading behavior of participants; however, the linearity of source code is a major driving factor determining programmers' reading order, even more important than experience.

Developers renewed focus on the last segment of the repeated code

According to Jbara and Feitelson [43], when in repeated segments, developers tend to invest more effort on the initial repetitions, and less on successive ones. They also renewed focus on the last segment of the function which is not part of the repetitive segments.

Indentation is simply a matter of task and style

Bauer et al. [46] conducted a non-exact replication of a Miara et al. [68] previous study using Java code snippets with varying indentation levels, and did not find any effect of indentation depth on program comprehension, perceived difficulty, or visual effort.

Developers focus more on code that follows readability rules

Peterson et al. [44] showed two versions of the same code, one following the readability rule and one not. When asked to choose, developers focused more on code snippets that followed readability rules, especially for avoiding nested if statements, showing their preference. Specifically, minimizing nesting and avoiding do-while loops.

Debugging time increases with variability in code

Santos and Sant' Anna [59] shows that comprehensibility was more negatively affected when a variable which is shared between features was defined in a point far from where it was used. Melo et al. [58] found that for data flow bugs, it was more helpful to dedicate attention to how the data values change during debugging. And for control flow bugs, to dedicate attention at the source code to understand the logic structure.

Method syntax is more difficult than Query Syntax

Katona et al. [48] evaluated Microsoft's statement that LINQ query syntax-based queries are easier for people to read and interpret than method syntax-based ones. It found that for method syntax, the information retrieval was more difficult and less efficient, requiring more mental effort to determine the query results.

3.4 RQ4. IN WHAT WAYS HAVE RESEARCHERS USED EYE-TRACKERS TO COLLECT AND VISUALIZE QUANTITATIVE MEASUREMENTS IN RECENT STUDIES?

3.4.1 Eye-tracking metrics

In these studies, researchers collect eye-tracking metrics, as described in the methods section, to provide valuable insights into visual attention and cognitive load during tasks.

To understand visual attention:

- Fixations: Indicates where the user is focusing their attention for extended periods.
- Fixation sequences: Indicates how information is processed and the cognitive strategy used
- Fixation Duration: Reveals how long the user remains focused on a specific area (AOI).
- Time to First Fixation: Indicates how quickly an element in a visual display captures the user's attention.
- Heat Maps: Shows the distribution of visual attention across the screen. Areas with more attention will be highlighted in "hotter"colors.
- Saccades: Quick and efficient saccades may suggest a smooth reading process, while long or irregular saccades might indicate difficulty in processing information.
- Regression: indicating re-reading or reevaluation behavior.

To understand Cognitive Load and Mental Effort:

- Pupil Size: Variations in pupil size can suggest cognitive load and mental effort. Larger pupils often indicate greater mental processing.
- Blinks: Increased blink frequency can be a sign of mental strain or stress.
- Fixation sequences: Complex sequences may indicate a higher cognitive load as users shift attention to process different areas.

As shown in Table 1, the most frequently reported being fixations, fixation duration, and fixation sequences.

Tabela 1 – Eye-tracking metrics used in the reviewed studies

Metric	Count
Fixations	16
Fixation duration	14
Fixation sequences	9
Saccades	8
Heat maps	5
Pupil dilation	4
Regression	3
Time to First Fixation (TFF)	1
Perceptual span	1
Blink rate	1

3.4.2 Experiment design

Across the debugging studies, participants were asked to examine source code to identify programming errors, although specific tasks varied according to each research design. While not all studies included every step shown, the flowchart (Fig. 5) summarizes the core activities participants engaged in across experiments.

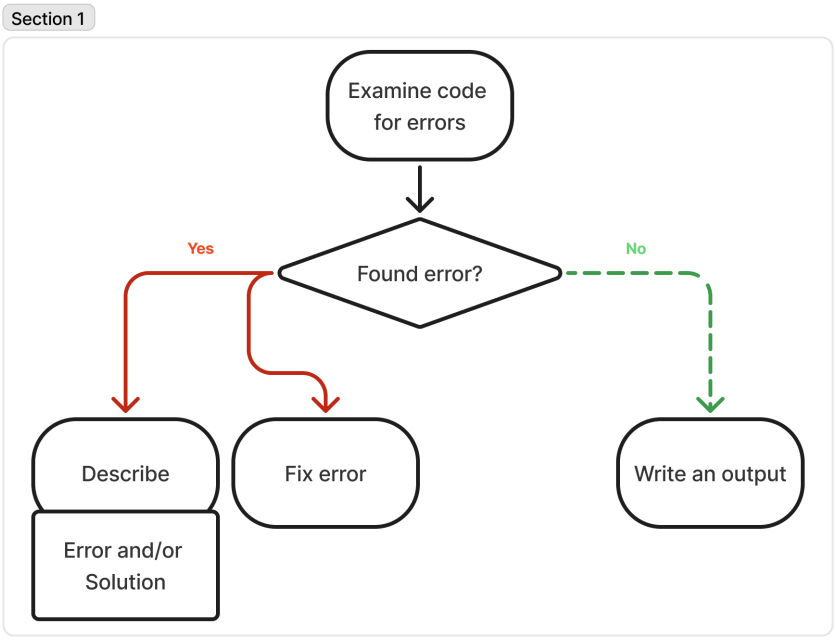

Figura 5 – Flowchart summarizing developer actions during debugging tasks in selected studies.

- In [Barik et al., 2017] [56], the task was time-constrained, giving developers five minutes

to identify compiler-related issues.

- In [Katona et al., 2019] [67], participants had to debug an insertion sort algorithm with hidden errors (Incorrect Loop Condition, Premature Loop Termination and Incorrect Index Assignment).
- In [Nivala et al., 2016] [60], participants were given eight C code snippets—five with common bugs such as surface-level errors, control flow issues, plan structure bugs, and structure interaction problems—and were asked to identify and describe errors, or write the expected output if none were found.
- In [Peng et al., 2016] [57], participants had to debug a heap sort algorithm with control flow bugs, and a quick sort algorithm with data flow bugs in an IDE with marked interest areas: debug thread, variables, code, outline, and console.
- In [Melo et al., 2017] [58] and Santos and Sant' Anna [59], participants were briefly trained on variability and feature-related concepts before identifying bugs tied to feature dependencies. In [Melo et al., 2017] [58], this task was followed by interviews in which participants reflected on their debugging experiences and strategies.
- In [Lin et al., 2016] [61], participants had to debug two programs (including iterative and recursive structures) while their performance was graded. The researchers used this grade to divided participants into high and low-performing groups based on debugging efficiency.
- In [Chandrika et al., 2017] [66], participants were asked to review source code with the intention of findings bugs, and talk aloud the error line and description.

Across the code comprehension studies, participants were asked to read source code and perform tasks such as summarizing the code, predicting program outputs, answering comprehension questions, or comparing and evaluating alternative code versions. While not all studies included every step shown, the flowchart (Fig. 6) summarizes the core activities participants engaged in across experiments.

- In Jbara and Feitelson [43], researchers followed a between-subject design to assign developers tasks with different versions of a program. Participants were asked if they understood what the program did, and to evaluate the code difficulty on a 5-point scale along with the evaluation reason.

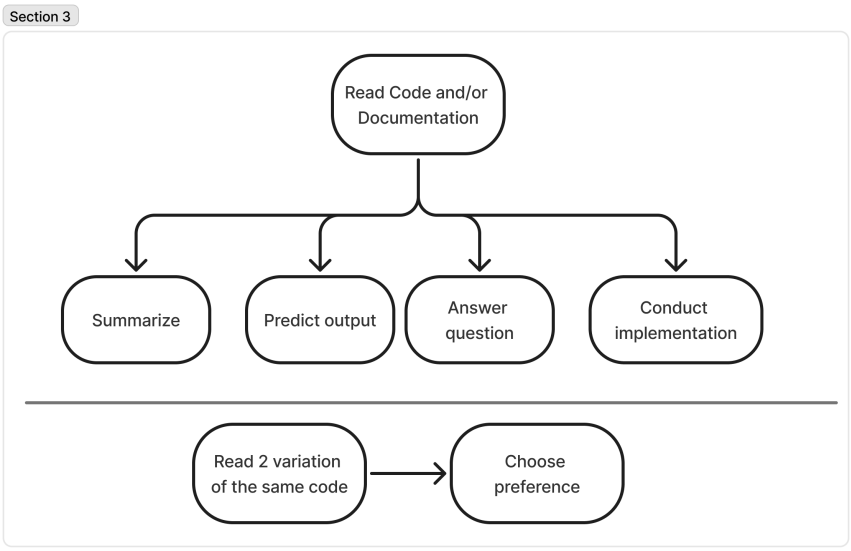

Figura 6 – Flowchart summarizing developer actions during code comprehension tasks in selected studies.

- In Peterson et al. [44], participants were shown two versions of the same code, one following the readability rule and one not, and had to choose between the two and justify their preference.
- In McChesney and Bond [47], participants had to read and understand Java programs (presented in either crowded or spaced formats), then describe the function of each program verbally, rate their own confidence in understanding, and were also evaluated by researchers on their comprehension.
- Participants had to create summaries explaining the code functionality or usage:
  - In Rodeghero et al. [64], participants had to read Java methods and write a summary for each one using a custom interface during a controlled, one-hour session.
  - In Peterson et al. [42], participants were asked to create summaries describing the implementation and usage of two methods and two classes, using both the Java source code and Stack Overflow (starting from the homepage) to search for relevant information.
  - In Abid et al. [65], developers had to read methods from a pool of 63 Java methods of 5 different open source programs, and write a summary for them.
  - In Saddler et al. [53], participants were tasked with summarizing four API elements — two methods and two classes — using Stack Overflow resources, without access to the source code.

- Participants had to answer comprehension questions:
  - In Peachock et al. [40], participants read three natural language texts and seven small C++ programs, then answered randomized comprehension questions (summary, multiple choice, or factual) and self-assessed difficulty.
  - In Busjahn et al. [52], developers were asked one of three possible questions: (1) write a summary of the code, (2) write the value of a variable after program execution, or (3) answer a multiple-choice question about the program.
  - In Simhandl et al. [51], the participants had to read and complete sentences about the specification of the code.
- Participants had to predict the output of the code:
  - In Peitek et al. [45], researchers conducted a non-exact replication of a previous study comparing the reading order of novices and expert programmers. Participants had to read the code snippet and give the final output of the code.
  - In Peterson et al. [49], participants had to read the three programs and were assigned one of three comprehension questions about the program's output, a short answer, or a multiple choice question.
  - In Bauer et al. [46], researchers conducted a non-exact replication of a previous study using Java code snippets with varying indentation levels. Participants had five minutes to read the code snippet and give the final output of the code. At the end, participants rated the difficult of the code snippets.
  - In Beelders [55], developers had to read a short code snippet from four programs with similar complexity, size, and structures, but with different indentation levels applied.
- Participants had up to a minute to observe each query.
- Participants read code/documentation and performed implementation-based tasks:
  - In Peterson [50], participants had to solve real database manipulation tasks using different programming language variants.
  - In Gorski et al. [54], participants had to perform a real-world integration task with differing documentation, integrating the Google Maps API and configuring a CSP policy using varied documentation.

Across the model comprehension studies, participants were asked to comprehend and analyze visual and textual abstractions:

- In Andrzejewska and Stolińska [62], participants had to solve algorithmic tasks presented as pseudocode and structured flowcharts, focusing on true-false and output-answer assignments.
- In Störrle et al. [63], users were presented various UML diagrams to analyze.

Related to traceability, in Kevic et al. [41], to investigate developers' detailed behavior while performing a change task, the participants had to work on three bugs for a total of 60 minutes.

3.4.3 Artifacts

Among the various artifacts analyzed, Java was the most widely used, appearing across multiple studies and topic domains (see Fig. 7). In contrast, studies focusing on model comprehension often relied on more abstract or visual artifacts, such as UML diagrams, flowcharts, or English pseudocode, which are more appropriate for representing high-level system models and design concepts.

Several of the top-used artifacts—Java, C++, C#, and JavaScript—share a common trait: they are object-oriented programming languages. This preference is likely influenced by both the structure and reusability offered by these languages, as well as the participants' familiarity. As illustrated in Fig. 8, studies focused particularly on students who had prior exposure to such languagues through academic programming courses.

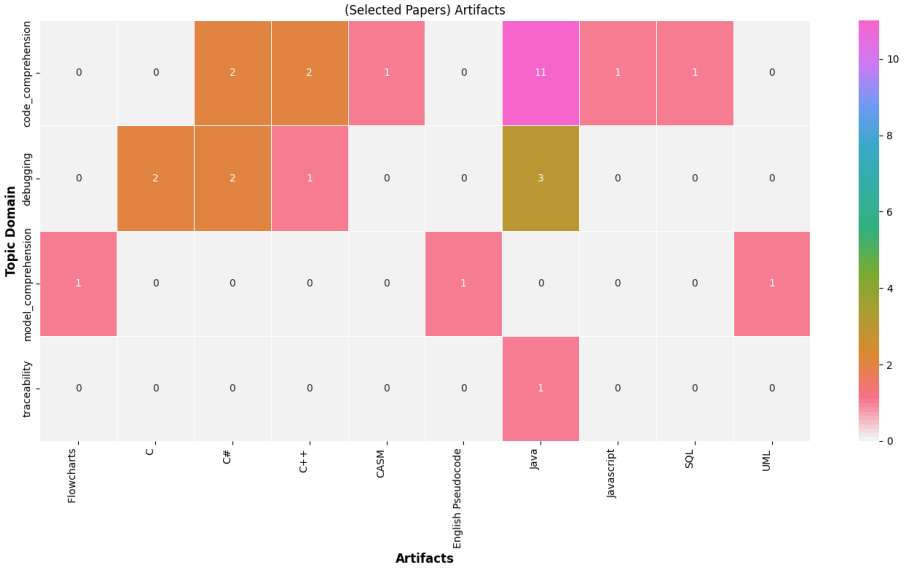

Figura 7 – The type of artifact used, grouped by Topic Domain.

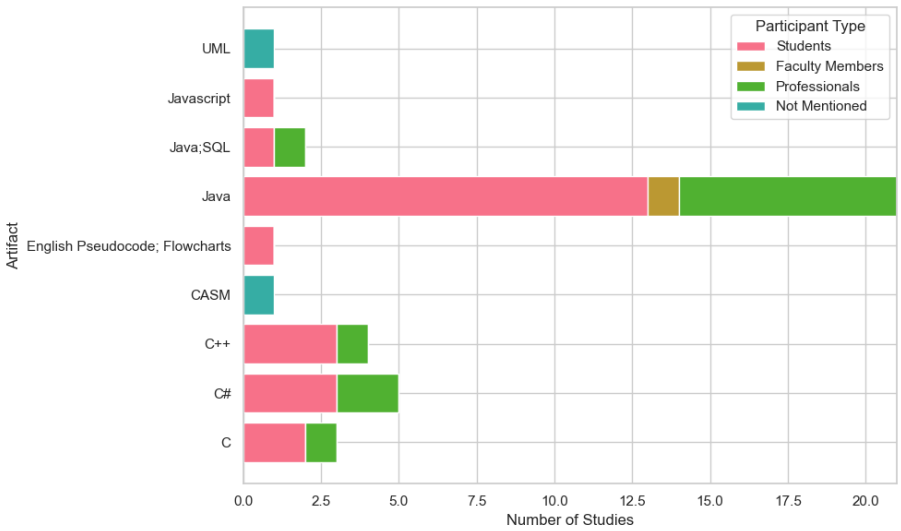

Figura 8 – Distribution of Programming Artifacts Across Participant Types in Research Studies

3.5 RQ5: WHAT ARE THE KEY LIMITATIONS IDENTIFIED IN RECENT EYE-TRACKING STUDIES WITHIN SOFTWARE ENGINEERING?

3.5.1 Participant Selection

Studies frequently had small sample sizes, lacked diversity, leading to questionable generalizability. As illustrated in Fig. 9, the majority involved between 20 and 30 participants. Larger studies were uncommon—only two included more than 50 participants, with the largest involving 114 participants [62]. Notably, one-third (33%) of the studies had fewer than 20

participants, with the smallest including just 9 participants [51].

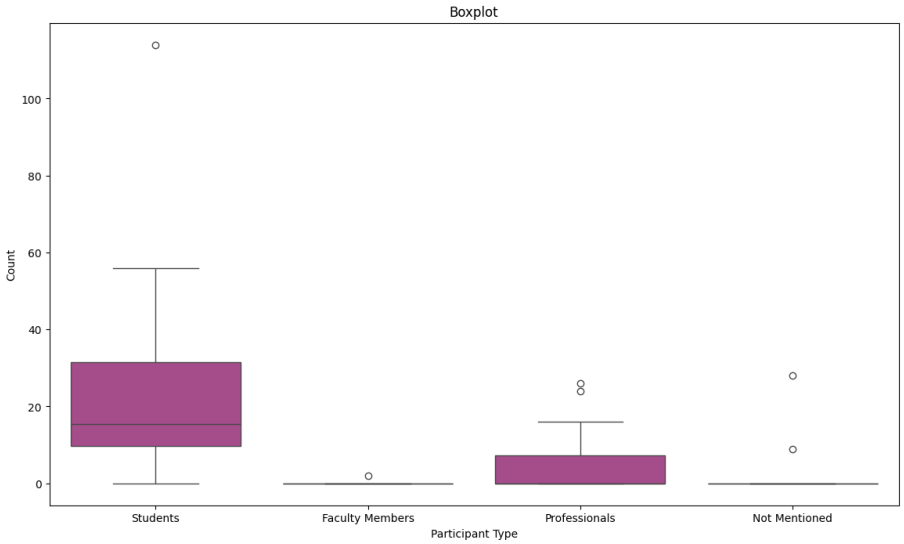

Figura 9 – Distribution of total participants across selected studies.

Figure 10 highlights a lack of diversity in participant expertise across the reviewed literature. Most studies primarily involved students, which appeared in 24 out of 28 studies. Professional participants were included in about 12 of those studies, although typically in smaller numbers compared to students. Only one study included faculty members, and two studies did not report participant backgrounds at all.

In the majority of those studies, experts were defined as individuals who had already completed a bachelor's degree and/or is a professional with experience in the field. Peachock et al. [40] was an exception, categorizing senior computer science students as experts, causing difficulties in detecting nuanced differences between novices and experts.

The language background or familiarity of the participants with the chosen Integrated Development Environment (IDE) might have influenced the comprehension and introduced an additional cognitive load. In Gorski et al [54], it was observed that developers mainly focus on code examples and skim through the rest of the documentation. However, in this study, the participants were German native speakers reading English documentation, which may have influenced comprehension and introduced an additional cognitive load.

3.5.2 Technical limitations with Eye-Tracker technology

Eye-tracking data was often incomplete due to equipment limitations or restricted tracking environments.

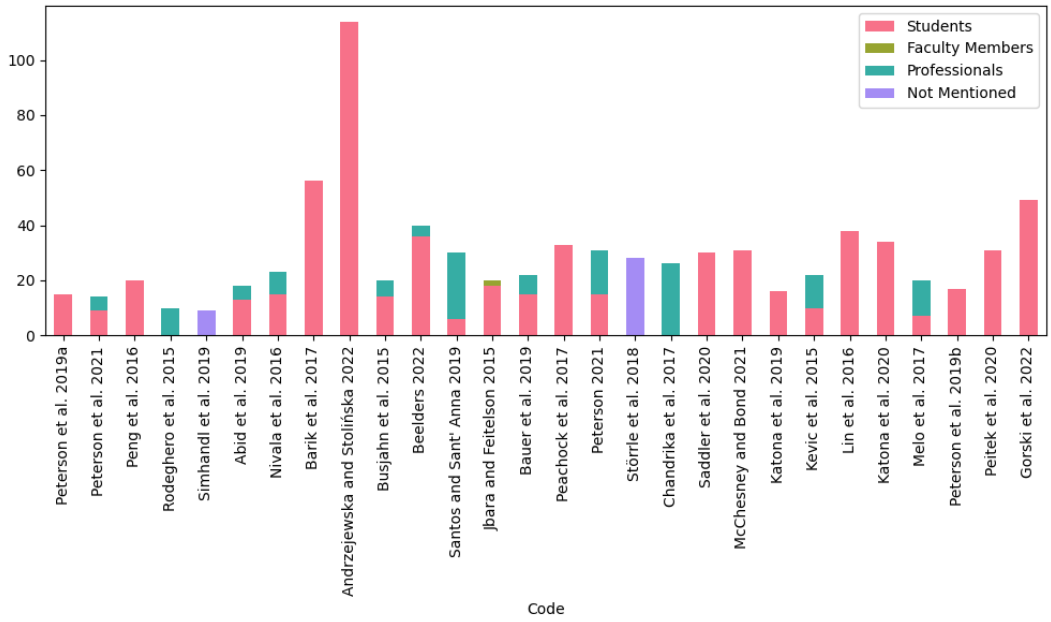

Figura 10 – Composition of study participants by type across reviewed studies.

Commodity eye trackers had lower sampling rates, which prevented detailed line-level or word-level analyses. Those eye-trackers struggled with accurately mapping gaze data to semantic code elements during dynamic contexts like actual code editing or writing. Studies commonly select small, non-scrollable code snippets or restricted participants from performing realistic actions such as freely browsing code or using external search engines, significantly limiting generalizability.

Eye-trackers also struggled with participants wearing glasses, and data quality was often compromised, resulting in discarded datasets. Some studies performed an occasionally necessary manual correction of fixation data, introducing potential bias and inaccuracies.

3.6 RQ6: WHAT EYE-TRACKERS ARE MOST FREQUENTLY USED IN RECENT EYE-TRACKING STUDIES?

In the reviewed studies, non-intrusive trackers were predominantly used. The screen-based (remote) eye-trackers do not require the participant to wear any equipment, and the tracker records the eye movements at a distance while mounted below or placed close to a computer or screen. Intrusive eye-trackers are mounted onto lightweight eyeglass frames, and record eye activity from a close range. Only Simhandl et al [51] used such type of tracker.

As shown in Figure 11, Tobii emerged as the most popular manufacturer for tracking

visual attention of participant in relation to programming. Tobii's models were used in 14 studies, including the X60, TX300, EyeX, and X3-120 models. SMI appears as the second most common, featured in 5 studies, employing models such as iViewX, REDm, RED 4, and RED Professional; GazePoint GP3 was reported in 3 studies; and EyeTribe, Eyelink, and Pupil Labs with one study each. Three studies did not specify which device was used.

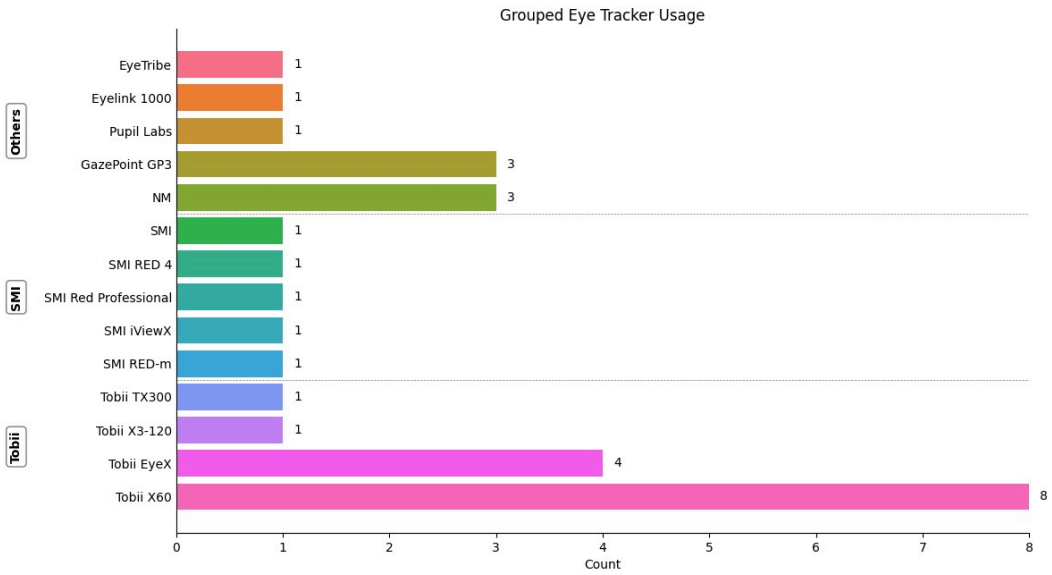

Figura 11 – The version of eye-tracker used, grouped by manufacturer

Several eye-trackers have been discontinued: EyeTribe ceased operations following its acquisition by Meta in 2016 [70]; SMI ceased all eye-tracking products and support following its acquisition by Apple in 2017 [69]; Tobii models such as the X60, TX300, EyeX, and X3-120 are no longer manufactured and have been succeeded by newer devices.

3.7 RQ7: HOW COULD EYE-TRACKING METHODOLOGY BE USED TO ANALYZE THE HARMFULNESS OF CODE SMELLS?

Martin Fowler defined "Refactoring"as "a change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior"(M. Fowler. Refactoring: Improving the design of existing code, 2018) [2]. Thus, a refactored code leads to savings in terms of time and resources. Understanding the harmfulness of code smells could assist programmers in identifying opportune moments for refactoring.

To harmfulness of code smells, in Lima et al. [72], it was adopted the categories CLEAN, SMELLY, BUGGY, and HARMFUL:

- CLEAN: code with no smells or bugs
- SMELLY: code with smells
- BUGGY: code with one or more defects or bugs that have been reported
- HARMFUL: a SMELLY code element having one or more bugs reported

3.7.1 Hypothesis

Debugging harmful code requires more cognitive effort for error-finding compared to debugging buggy code.

3.7.2 Procedure

Use eye-tracking technology to measure visual attention and cognitive load while participants look for errors in different snippets of code. Compare the performance and correctness across the buggy and harmful programs.

3.7.3 Artifacts

Buggy and Harmful code snippets in Python language, due to its popularity and simplicity for novices, especially given the impact of language familiarity as a limitation commonly noted on this study's analyzed papers.

3.7.4 Study Variables

These metrics will be obtained through the use of eye-tracker to support the experiment:

- Fixation sequences: to understand participants' cognitive behavior while looking at the segment containing an error, as well as shifting attention to areas with smells
- Fixation Duration: to measure the complexity of understanding and fixing the error
- Time to First Fixation: to quantify how long the participant takes to find the error

- Heat Maps: to understand visual focus on the smell and how it impacts the time spent on error finding
- Saccades: to identify the reading process while switching between the smelly and normal code
- Pupil Size and Blinks: to measure the cognitive effort while being exposed to the smell

Additionally, we will consider whether the participant locates the bug and the time it took.

3.7.5 Participants

Based on the documents reviewed as part of this updated SLR, it is possible to notice that the majority of the studies use a sample of 30 participants or less. In order to abide to the standard set by these previous researches, it is proposed that future studies use a diverse group with at least 40 participants, 20 novices and 20 experts.

4 CONCLUSION

The systematic literature review presented here reaffirms eye-tracking technology as an effective means of investigating developer interactions and cognitive processes related to software engineering activities. The review shows advancements in research methodologies since 2015 and limitations such as small participant groups, limited diversity, and technological constraints of eye-tracking equipment. Recognizing these insights, an eye-tracking experiment targeting the assessment of code smells' harmfulness was proposed, through measurable cognitive load and debugging performance metrics. Future work will focus on executing this proposed experimental setup to empirically understand the impact of code smells, offering practical implications for code refactoring practices. Additionally, further enhancements to the Review Wise tool can expand its capabilities across additional phases of systematic literature reviews.

REFERÊNCIAS

"Exploring Software Development with Eye Tracking: An Interview with Dr. Bonita Sharif." Tobii, https://www.tobii.com/resource-center/research-spotlight-interviews/exploring-software-development-with-eye-tracking

M. Fowler (2018). Refactoring: Improving the design of existing code. Addison-Wesley.

Sharafi, Zohreh and Soh, Zephyrin and Gueh'eneuc, Yann-Ga. "A systematic literature review on the usage of eye-tracking in software engineering", in Inf. Softw. Technol., 2015, pp. 79–107, doi: 10.1016/j.infsof.2015.06.008

Bryn Farnsworth (2022). Eye Tracking: The Complete Pocket Guide. iMotions. https://imotions.com/blog/learning/best-practice/eye-tracking/

D. G. Feitelson, "Eye Tracking and Program Comprehension", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 1-1, doi: 10.1109/EMIP.2019.00008

N. Mansoor, "Empirical Assessment of Program Comprehension Styles in Programming Language Paradigms", in 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2021, pp. 1-2, doi: 10.1109/VL/HCC51201.2021.9576333

Q. Mi, J. Keung, J. Huang, and Y. Xiao, "Using Eye Tracking Technology to Analyze the Impact of Stylistic Inconsistency on Code Readability", in 2017 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C), 2017, pp. 579-580, doi: 10.1109/QRS-C.2017.102

U. Obaidellah, A. L. M. Haek, and P. C. Cheng, "A Survey on the Usage of Eye-Tracking in Computer Programming", in Unknown Conference, 2018, pp. xx-xx, doi: 10.1145/3145904

K. Kevic, "Recognizing Relevant Code Elements during Change Task Navigation", in 2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C), 2016, pp. 851-854, doi: Unknown DOI

S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The Effect of Poor Source Code Lexicon and Readability on Developers' Cognitive Load", in 2018 IEEE/ACM 26th

International Conference on Program Comprehension (ICPC), 2018, pp. 286-28610, doi: Unknown DOI

A. Abbad-Andaloussi, T. Sorg, and B. Weber, "Estimating Developers' Cognitive Load at a Fine-grained Level Using Eye-tracking Measures", in 2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC), 2022, pp. 111-121, doi: 10.1145/3524610.3527890

M. Ahrens, K. Schneider, and M. Busch, "Attention in Software Maintenance: An Eye Tracking Study", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 2-9, doi: 10.1109/EMIP.2019.00009

Z. Ahsan, and U. Obaidellah, "Predicting expertise among novice programmers with prior knowledge on programming tasks", in 2020 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC), 2020, pp. 1008-1016, doi: Unknown DOI

J. Behler, P. Weston, D. T. Guarnera, B. Sharif, and J. I. Maletic, "iTrace-Toolkit: A Pipeline for Analyzing Eye-Tracking Data of Software Engineering Studies", in 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2023, pp. 46-50, doi: 10.1109/ICSE-Companion58688.2023.00022

B. Clark, and B. Sharif, "iTraceVis: Visualizing Eye Movement Data Within Eclipse", in 2017 IEEE Working Conference on Software Visualization (VISSOFT), 2017, pp. 22-32, doi: 10.1109/VISSOFT.2017.30

F. Deitelhoff, and A. Harrer, "Towards a Dynamic Help System: Support of Learners During Programming Tasks Based Upon Historical Eye-Tracking Data", in 2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT), 2018, pp. 77-78, doi: 10.1109/ICALT.2018.00116

F. Deitelhoff, A. Harrer, and A. Kienle, "The Influence of Different AOI Models in Source Code Comprehension Analysis", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 10-17, doi: 10.1109/EMIP.2019.00010

S. Fakhoury, D. Roy, H. Pines, T. Cleveland, C. S. Peterson, V. Arnaoudova, B. Sharif, and J. Maletic, "gazel: Supporting Source Code Edits in Eye-Tracking Studies", in 2021

IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2021, pp. 69-72, doi: 10.1109/ICSE-Companion52605.2021.00038

H. Hijazi, J. Duraes, R. Couceiro, J. Castelhano, R. Barbosa, J. Medeiros, M. Castelo-Branco, P. de Carvalho, and H. Madeira, "Quality Evaluation of Modern Code Reviews Through Intelligent Biometric Program Comprehension", in IEEE Transactions on Software Engineering, 2023, pp. 626-645, doi: 10.1109/TSE.2022.3158543

C. Ioannou, P. Bækgaard, E. Kindler, and B. Weber, "Towards a tool for visualizing pupil dilation linked with source code artifacts", in 2020 Working Conference on Software Visualization (VISSOFT), 2020, pp. 105-109, doi: 10.1109/VISSOFT51673.2020.00016

P. Jermann, and K. Sharma, "Gaze as a Proxy for Cognition and Communication", in 2018 IEEE 18th International Conference on Advanced Learning Technologies (ICALT), 2018, pp. 152-154, doi: 10.1109/ICALT.2018.00043

T. Kano, R. Sakagami, and T. Akakura, "Modeling of cognitive processes based on gaze transition during programming debugging", in 2021 IEEE 3rd Global Conference on Life Sciences and Technologies (LifeTech), 2021, pp. 412-413, doi: 10.1109/LifeTech52111.2021.9391940

J. Katona, A. Kovari, I. Heldal, C. Helgesen, C. Costescu, A. Rosan, A. Hathazi, S. Thill, and R. Demeter, "Recording Eye-tracking Parameters during a Program Source-code Debugging Example", in 2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2019, pp. 335-338, doi: 10.1109/CogInfoCom47531.2019.9089941

K. Kevic, "Using Eye Gaze Data to Recognize Task-Relevant Source Code Better and More Fine-Grained", in 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), 2017, pp. 103-105, doi: 10.1109/ICSE-C.2017.152

X. Li, W. Liu, H. Liu, J. Xu, and W. Cheng, "Task-oriented Analysis on Debugging Process Based on Eye Movements and IDE Interactions", in 2021 16th International Conference on Computer Science & Education (ICCSE), 2021, pp. 379-384, doi: 10.1109/ICCSE51940.2021.9569438

X. Li, W. Liu, W. Wang, J. Zhong, and M. Yu, "Assessing Students' Behavior in Error Finding Programming Tests: An Eye-Tracking Based Approach", in 2019 IEEE International Conference on Engineering, Technology and Education (TALE), 2019, pp. 1-6, doi: 10.1109/TALE48000.2019.9225906

L. Liu, W. Liu, X. Li, W. Wang, and W. Cheng, “Eye-tracking Based Performance Analysis in Error Finding Programming Test”, in 2020 15th International Conference on Computer Science & Education (ICCSE), 2020, pp. 477-482, doi: 10.1109/ICCSE49874.2020.9201882

N. Al Madi, C. S. Peterson, B. Sharif, and J. I. Maletic, “From Novice to Expert: Analysis of Token Level Effects in a Longitudinal Eye Tracking Study”, in 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2021, pp. 172-183, doi: 10.1109/ICPC52881.2021.00025

K. Mangaroska, K. Sharma, M. Giannakos, H. Trætteberg, and P. Dillenbourg, “Gaze Insights into Debugging Behavior Using Learner-Centred Analysis”, in Proceedings of the 8th International Conference on Learning Analytics and Knowledge, 2018, pp. 350-359, doi: 10.1145/3170358.3170386

J. Mucke, M. Schwarzkopf, and J. Siegmund, “REyeker: Remote Eye Tracker”, in ACM Symposium on Eye Tracking Research and Applications, 2021, pp. xx-xx, doi: 10.1145/3448018.3457423

D. Roy, S. Fakhoury, and V. Arnaoudova, “VITALSE: Visualizing Eye Tracking and Biometric Data”, in 2020 IEEE/ACM 42nd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2020, pp. 57-60, doi: Unknown DOI

T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif, “ITrace: Enabling Eye Tracking on Software Artifacts within the IDE to Support Software Engineering Tasks”, in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. 954-957, doi: 10.1145/2786805.2803188

Z. Sharafi, I. Bertram, M. Flanagan, and W. Weimer, “Eyes on Code: A Study on Developers’ Code Navigation Strategies”, in IEEE Transactions on Software Engineering, 2022, pp. 1692-1704, doi: 10.1109/TSE.2020.3032064

B. Sharif, A. Begel, and J. I. Maletic, “Conducting Eye Tracking Studies in Software Engineering - Methodology and Pipeline”, in 2023 IEEE/ACM 45th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion), 2023, pp. 340-341, doi: 10.1109/ICSE-Companion58688.2023.00097

B. Sharif, C. Peterson, D. Guarnera, C. Bryant, Z. Buchanan, V. Zyrianov, and J. Maletic, "Practical Eye Tracking with iTrace", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 41-42, doi: 10.1109/EMIP.2019.00015

B. Sharif, and J. I. Maletic, "iTrace: Overcoming the Limitations of Short Code Examples in Eye Tracking Experiments", in 2016 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2016, pp. 647-647, doi: 10.1109/ICSME.2016.61

T. Sorg, A. Abbad-Andaloussi, and B. Weber, "Towards a Fine-grained Analysis of Cognitive Load During Program Comprehension", in 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2022, pp. 748-752, doi: 10.1109/SANER53432.2022.00092

L. Zhang, J. Sun, C. Peterson, B. Sharif, and H. Yu, "Exploring Eye Tracking Data on Source Code via Dual Space Analysis", in 2019 Working Conference on Software Visualization (VISSOFT), 2019, pp. 67-77, doi: 10.1109/VISSOFT.2019.00016

V. Zyrianov, D. T. Guarnera, C. S. Peterson, B. Sharif, and J. I. Maletic, "Automated Recording and Semantics-Aware Replaying of High-Speed Eye Tracking and Interaction Data to Support Cognitive Studies of Software Engineering Tasks", in 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2020, pp. 464-475, doi: 10.1109/ICSME46990.2020.00051

P. Peachock, N. Iovino, and B. Sharif, "Investigating Eye Movements in Natural Language and C++ Source Code - A Replication Experiment", in Augmented Cognition. Neurocognition and Machine Learning, 2017, pp. xx-xx, doi: 10.1007/978-3-319-58628-1_17

K. Kevic, B. Walters, T. Shaffer, B. Sharif, D. Shepherd, and T. Fritz, "Tracing software developers' eyes and interactions for change tasks", in Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, 2015, pp. xx-xx, doi: 10.1145/2786805.2786864

C. Peterson, J. Saddler, N. Halavick, and B. Sharif, "A Gaze-Based Exploratory Study on the Information Seeking Behavior of Developers on Stack Overflow", in Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems, 2019, pp. xx-xx, doi: 10.1145/3290607.3312801

A. Jbara, and D. G. Feitelson, "How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking", in 2015 IEEE 23rd International Conference on Program Comprehension, 2015, pp. 244-254, doi: 10.1109/ICPC.2015.35

C. S. Peterson, K. -i. Park, I. Baysinger, and B. Sharif, "An Eye Tracking Perspective on How Developers Rate Source Code Readability Rules", in 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW), 2021, pp. 138-139, doi: 10.1109/ASEW52652.2021.00037

N. Peitek, J. Siegmund, and S. Apel, "What Drives the Reading Order of Programmers? An Eye Tracking Study", in 2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC), 2020, pp. 342-353, doi: 10.1145/3387904.3389279

J. Bauer, J. Siegmund, N. Peitek, J. C. Hofmeister, and S. Apel, "Indentation: Simply a Matter of Style or Support for Program Comprehension?", in 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 154-164, doi: 10.1109/ICPC.2019.00033

I. McChesney, and R. Bond, "The Effect Of Crowding On The Reading Of Program Code For Programmers With Dyslexia", in 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC), 2021, pp. 300-310, doi: 10.1109/ICPC52881.2021.00036

J. Katona, A. Kovari, I. Heldal, C. Costescu, A. Rosan, R. Demeter, S. Thill, and T. Stefanut, "Using Eye- Tracking to Examine Query Syntax and Method Syntax Comprehension in LINQ", in 2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2020, pp. 000437-000444, doi: 10.1109/CogInfoCom50765.2020.9237910

C. S. Peterson, J. A. Saddler, T. Blascheck, and B. Sharif, "Visually Analyzing Students' Gaze on C++ Code Snippets", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 18-25, doi: 10.1109/EMIP.2019.00011

C. S. Peterson, "Investigating the Effect of Polyglot Programming on Developers", in 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2021, pp. 1-2, doi: 10.1109/VL/HCC51201.2021.9576404

G. Simhandl, P. Paulweber, and U. Zdun, "Design of an Executable Specification Language Using Eye Tracking", in 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP), 2019, pp. 37-40, doi: 10.1109/EMIP.2019.00014

T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, "Eye Movements in Code Reading: Relaxing the Linear Order", in 2015 IEEE 23rd International Conference on Program Comprehension, 2015, pp. 255-265, doi: 10.1109/ICPC.2015.36

J. A. Saddler, C. S. Peterson, S. Sama, S. Nagaraj, O. Baysal, L. Guerrouj, and B. Sharif, "Studying Developer Reading Behavior on Stack Overflow during API Summarization Tasks", in 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 195-205, doi: 10.1109/SANER48275.2020.9054848

P. L. Gorski, S. Möller, S. Wiefling, and L. L. Iacono, ""I just looked for the solution!"On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices", in IEEE Transactions on Software Engineering, 2022, pp. 3467-3484, doi: 10.1109/TSE.2021.3094171

T. Beelders, "Eye-tracking analysis of source code reading on a line-by-line basis", in 2022 IEEE/ACM 10th International Workshop on Eye Movements in Programming (EMIP), 2022, pp. 1-7, doi: Unknown DOI

T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin, "Do Developers Read Compiler Error Messages?", in 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), 2017, pp. 575-585, doi: 10.1109/ICSE.2017.59

F. Peng, C. Li, X. Song, W. Hu, and G. Feng, "An Eye Tracking Research on Debugging Strategies towards Different Types of Bugs", in 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2016, pp. 130-134, doi: 10.1109/COMPSAC.2016.57

J. Melo, F. B. Narcizo, D. W. Hansen, C. Brabrand, and A. Wasowski, "Variability through the Eyes of the Programmer", in 2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC), 2017, pp. 34-44, doi: 10.1109/ICPC.2017.34

D. Santos, and C. Sant' Anna, "How Does Feature Dependency Affect Configurable System Comprehensibility?", in 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC), 2019, pp. 19-29, doi: 10.1109/ICPC.2019.00016

M. Nivala, F. Hauser, J. Mottok, and H. Gruber, "Developing visual expertise in software engineering: An eye tracking study", in 2016 IEEE Global Engineering Education Conference (EDUCON), 2016, pp. 613-620, doi: 10.1109/EDUCON.2016.7474614

Y. -T. Lin, C. -C. Wu, T. -Y. Hou, Y. -C. Lin, F. -Y. Yang, and C. -H. Chang, "Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach", in IEEE Transactions on Education, 2016, pp. 175-186, doi: 10.1109/TE.2015.2487341

M. Andrzejewska, and A. Stolińska, "Do Structured Flowcharts Outperform Pseudocode? Evidence From Eye Movements", in IEEE Access, 2022, pp. 132965-132975, doi: 10.1109/ACCESS.2022.3230981

H. Störrle, N. Baltsen, H. Christoffersen, and A. M. Maier, "Poster: How Do Modelers Read UML Diagrams? Preliminary Results from an Eye-Tracking Study", in 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion), 2018, pp. 396-397, doi: Unknown DOI

P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization", in IEEE Transactions on Software Engineering, 2015, pp. 1038-1054, doi: 10.1109/TSE.2015.2442238

N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic, "Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters", in 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), 2019, pp. 384-395, doi: 10.1109/ICSE.2019.00052

K. R. Chandrika, J. Amudha, and S. D. Sudarsan, "Recognizing eye tracking traits for source code review", in 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), 2017, pp. 1-8, doi: 10.1109/ETFA.2017.8247637

J. Katona, A. Kovari, C. Costescu, A. Rosan, A. Hathazi, I. Heldal, C. Helgesen, S. Thill, and R. Demeter, "The Examination Task of Source-code Debugging Using GP3 Eye Tracker", in 2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom), 2019, pp. 329-334, doi: 10.1109/CogInfoCom47531.2019.9089952

R. J. Miara, J. A. Musselman, J. A. Navarro, and B. Shneiderman, "Program indentation and comprehensibility," Communications of the ACM, vol. 26, no. 11, pp. 861–867, 1983.

Matney, L. "Apple Acquires SMI Eye-Tracking Company."TechCrunch, 26 June 2017, https://techcrunch.com/2017/06/26/apple-acquires-smi-eye-tracking-company/.

Constine, J. "Oculus Acquires Eye-Tracking Startup The Eye Tribe."TechCrunch, 28 December 2016, https://techcrunch.com/2017/06/26/apple-acquires-smi-eye-tracking-company/.

Korbel, J.O., Stegle, O. Effects of the COVID-19 pandemic on life scientists. Genome Biol 21, 113 (2020). https://doi.org/10.1186/s13059-020-02031-1

Rodrigo Lima, Jairo Souza, Baldoino Fonseca, Leopoldo Teixeira, Rohit Gheyi, Márcio Ribeiro, Alessandro Garcia, and Rafael de Mello. 2020. Understanding and Detecting Harmful Code. In Proceedings of the XXXIV Brazilian Symposium on Software Engineering (SBES '20). Association for Computing Machinery

.1 SUMMARY OF SELECTED STUDIES

[Peterson et al, 2019a] [42]

C. Peterson, J. Saddler, N. Halavick, and B. Sharif (2019). A Gaze-Based Exploratory Study on the Information Seeking Behavior of Developers on Stack Overflow. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems"

Eye-tracker Tobii X-60

Artifacts Java

Participants 15 Students

[Peterson et al, 2021] [44]

C. S. Peterson, K. -i. Park, I. Baysinger, and B. Sharif (2021). An Eye Tracking Perspective on How Developers Rate Source Code Readability Rules. In 2021 36th IEEE/ACM International Conference on Automated Software Engineering Workshops (ASEW)"

Eye-tracker Tobii X60

Artifacts Java

Participants 14 Students and Professionals

[Peng et al, 2016] [57]

F. Peng, C. Li, X. Song, W. Hu, and G. Feng (2016). An Eye Tracking Research on Debugging Strategies towards Different Types of Bugs. In 2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)"

Eye-tracker Tobii X60

Artifacts Java

Participants 20 Students

[Rodeghero et al, 2015] [64]

P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan (2015). An Eye-Tracking Study of Java Programmers and Application to Source Code Summarization. In IEEE Transactions on

Software Engineering"

Eye-tracker Tobii TX300

Artifacts Java

Participants 10 Professionals

[Simhandl et al, 2019] [51]

G. Simhandl, P. Paulweber, and U. Zdun (2019). Design of an Executable Specification Language Using Eye Tracking. In 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)"

Eye-tracker Pupil Labs

Artifacts CASM

Participants 9 Not Mentioned

[Abid et al, 2019] [65]

N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic (2019). Developer Reading Behavior While Summarizing Java Methods: Size and Context Matters. In 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)"

Eye-tracker Tobii X60

Artifacts Java

Participants 18 Students and Professionals

[Nivala et al, 2016] [60]

M. Nivala, F. Hauser, J. Mottok, and H. Gruber (2016). Developing visual expertise in software engineering: An eye tracking study. In 2016 IEEE Global Engineering Education Conference (EDUCON)"

Eye-tracker SMI

Artifacts C

Participants 23 Students;Professionals

[Barik et al, 2017] [56]

T. Barik, J. Smith, K. Lubick, E. Holmes, J. Feng, E. Murphy-Hill, and C. Parnin (2017). Do Developers Read Compiler Error Messages?. In 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)"

Eye-tracker GazePoint GP3

Artifacts Java

Participants 56 Students

[Andrzejewska and Stolińska, 2022] [62]

M. Andrzejewska, and A. Stolińska (2022). Do Structured Flowcharts Outperform Pseudocode? Evidence From Eye Movements. In IEEE Access"

Eye-tracker SMI iViewX

Artifacts English Pseudocode; Flowcharts

Participants 114 Students

[Busjahn et al, 2015] [52]

T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm (2015). Eye Movements in Code Reading: Relaxing the Linear Order. In 2015 IEEE 23rd International Conference on Program Comprehension"

Eye-tracker SMI REDm

Artifacts Java

Participants 20 Students and Professionals

[Beelders, 2022] [55]

T. Beelders (2022). Eye-tracking analysis of source code reading on a line-by-line basis. In 2022 IEEE/ACM 10th International Workshop on Eye Movements in Programming (EMIP)

Eye-tracker Not Mentioned

Artifacts C#

Participants 40 Students and Professionals

[Santos and Sant' Anna, 2019] [59]

D. Santos, and C. Sant' Anna (2019). How Does Feature Dependency Affect Configurable System Comprehensibility?. In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)"

Eye-tracker Tobii EyeX

Artifacts C++

Participants 30 Students and Professionals

[Jbara and Feitelson, 2015] [43]

A. Jbara, and D. G. Feitelson (2015). How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking. In 2015 IEEE 23rd International Conference on Program Comprehension"

Eye-tracker EyeTribe

Artifacts Java

Participants 20 Students;Faculty Members

[Bauer et al, 2019] [46]

J. Bauer, J. Siegmund, N. Peitek, J. C. Hofmeister, and S. Apel (2019). Indentation: Simply a Matter of Style or Support for Program Comprehension?. In 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)"

Eye-tracker Tobii EyeX

Artifacts Java

Participants 22 Students and Professionals

[Peachock et al, 2017] [40]

P. Peachock, N. Iovino, and B. Sharif (2017). Investigating Eye Movements in Natural Language and C++ Source Code - A Replication Experiment. In Augmented Cognition. Neurocognition and Machine Learning"

Eye-tracker Tobii X60

Artifacts C++

Participants 33 Students

[Peterson, 2021] [50]

C. S. Peterson (2021). Investigating the Effect of Polyglot Programming on Developers. In 2021 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)

Eye-tracker Not Mentioned

Artifacts Java and SQL

Participants 31 Students and Professionals

[Störrle et al, 2018] [63]

H. Störrle, N. Baltsen, H. Christoffersen, and A. M. Maier (2018). Poster: How Do Modelers Read UML Diagrams? Preliminary Results from an Eye-Tracking Study. In 2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)"

Eye-tracker SMI RED 4

Artifacts UML

Participants 28 Not Mentioned

[Chandrika et al, 2017] [66]

K. R. Chandrika, J. Amudha, and S. D. Sudarsan (2017). Recognizing eye tracking traits for source code review. In 2017 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)"

Eye-tracker SMI Red Professional

Artifacts C#

Participants 26 Professionals

[Saddler et al, 2020] [53]

J. A. Saddler, C. S. Peterson, S. Sama, S. Nagaraj, O. Baysal, L. Guerrouj, and B. Sharif (2020). Studying Developer Reading Behavior on Stack Overflow during API Summarization Tasks. In 2020 IEEE 27th International Conference on Software Analysis, Evolution and Re-engineering (SANER)"

Eye-tracker Tobii X60

Artifacts Java

Participants 30 Students

[McChesney and Bond, 2021] [47]

I. McChesney, and R. Bond (2021). The Effect Of Crowding On The Reading Of Program Code For Programmers With Dyslexia. In 2021 IEEE/ACM 29th International Conference on Program Comprehension (ICPC)"

Eye-tracker Tobii X3-120

Artifacts Java

Participants 31 Students

[Katona et al, 2019] [67]

J. Katona, A. Kovari, C. Costescu, A. Rosan, A. Hathazi, I. Heldal, C. Helgesen, S. Thill, and R. Demeter (2019). The Examination Task of Source-code Debugging Using GP3 Eye Tracker. In 2019 10th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)"

Eye-tracker GazePoint GP3

Artifacts C#

Participants 16 Students

[Kevic et al, 2015] [41]

K. Kevic, B. Walters, T. Shaffer, B. Sharif, D. Shepherd, and T. Fritz (2015). Tracing software developers' eyes and interactions for change tasks. In Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering"

Eye-tracker Tobii X-60

Artifacts Java

Participants 22 Students;Professionals

[Lin et al, 2016] [61]

Y. -T. Lin, C. -C. Wu, T. -Y. Hou, Y. -C. Lin, F. -Y. Yang, and C. -H. Chang (2016). Tracking Students' Cognitive Processes During Program Debugging—An Eye-Movement Approach. In IEEE Transactions on Education"

Eye-tracker Eyelink 1000

Artifacts C

Participants 38 Students

[Katona et al, 2020] [48]

J. Katona, A. Kovari, I. Heldal, C. Costescu, A. Rosan, R. Demeter, S. Thill, and T. Stefanut (2020). Using Eye- Tracking to Examine Query Syntax and Method Syntax Comprehension in LINQ. In 2020 11th IEEE International Conference on Cognitive Infocommunications (CogInfoCom)"

Eye-tracker GazePoint GP3

Artifacts C

Participants 34 Students

[Melo et al., 2017] [58]

J. Melo, F. B. Narcizo, D. W. Hansen, C. Brabrand, and A. Wasowski (2017). Variability through the Eyes of the Programmer. In 2017 IEEE/ACM 25th International Conference on

Program Comprehension (ICPC)"

Eye-tracker Tobii EyeX

Artifacts Java

Participants 20 Students;Professionals

[Peterson et al., 2019b] [49]
C. S. Peterson, J. A. Saddler, T. Blascheck, and B. Sharif (2019). Visually Analyzing Students' Gaze on C++ Code Snippets. In 2019 IEEE/ACM 6th International Workshop on Eye Movements in Programming (EMIP)"

Eye-tracker Tobii X60

Artifacts C++

Participants 17 Students

[Peitek et al., 2020] [45]
N. Peitek, J. Siegmund, and S. Apel (2020). What Drives the Reading Order of Programmers? An Eye Tracking Study. In 2020 IEEE/ACM 28th International Conference on Program Comprehension (ICPC)"

Eye-tracker Tobii EyeX

Artifacts Java

Participants 31 Students

[Gorski et al., 2022] [54]
P. L. Gorski, S. Möller, S. Wiefling, and L. L. Iacono (2022). "I just looked for the solution!"On Integrating Security-Relevant Information in Non-Security API Documentation to Support Secure Coding Practices. In IEEE Transactions on Software Engineering"

Eye-tracker Not Mentioned

Artifacts Javascript

Participants 49 Students

.2 SUMMARY OF PAPERS REMOVED DURING ELIGIBILITY ASSESSMENT

- Fakhoury et al. [10] only employ eye-tracking as a supplementary to fNIRS and not the primary method of investigation. It investigated the impact of poor source code lexicon and readability on developers' cognitive load by combining fNIRS (functional Near Infrared Spectroscopy) brain imaging and eye-tracking data to map cognitive load to specific code elements.
- Studies that did not collect or analyze new empirical data:
  - Feitelson [5] was only a working session at a conference, not presenting original research;
  - Mansoor [6] and Mi et al. [7] only outline future research plans;
  - Obaidellah et al. [8] performed a survey that synthesizes and reviews existing literature;
  - Kevic [9] analyzed his previous research [41] that conducts an exploratory work using eye-tracking to create a model that can predict relevant code elements for better tool support; this was later included in the snowballing step.
- Studies that had other primary goals and did not focus directly on software engineering activities:
  - Propose a new approach to compute, represent, or predict visual attention patterns:
    - Jermann and Sharma [21] present a novel dialogue coding scheme that examines the relationship between gaze patterns, dialogue, and performance in spatially distributed (remote) pair programming tasks.
    - Zyrianov et al. [39], Zhang et al. [38], Sharafi et al. [33], and Kevic [24] present novel experimental designs to capture realistic software maintenance processes

in terms of software complexity and IDE interactions, allowing for scrolling, editing, and navigating large databases.

* Abbad-Andaloussi et al. [11] proposes an approach to estimate the mentally demanding parts of source code using eye-tracking and machine learning.
* Madi et al. [28] proposes a machine learning model that uses token frequency and token length in reading source code to estimate programming proficiency and classify novices and experts.
* Hijazi et al. [19] collects biometric data and uses Artificial Intelligence techniques to estimate the engagement and how well the reviewer has covered and understood the different regions of the code under review.
* Ahsan and Obaidellah [13] uses machine learning to predict expertise among novice programmers based on their performance and gaze metrics.
* Kano et al. [22] analyzes gaze transitions during programming debugging to create a transition diagram based on a Markov process and a Z-test to represent how gaze moves between different components and code blocks in a programming editor.
* Deitelhoff et al. [17] proposes one AOI model and compares the influence of those different AOI models from a methodical point of view.
* Sorg et al. [37] proposes a fine-grained analysis of cognitive load during program comprehension to identify critical parts of the code (PoCs).
* Ioannou et al. [20] proposes to create a tool that visualizes cognitive processes linked to specific source code artifacts. The paper uses eye-trackers to capture pupil dilation and fixation as indicators of cognitive load during code comprehension.
* To help teachers evaluate students' debugging skills and identify areas for improvement, Li et al. [26] proposes an eye-tracking based assessing framework for evaluating students' performance during error-finding tasks. Afterwards, Liu et al. [27] proposes a new metric, matched ratio of program execution, to describe the proportion of gaze path matching with program execution. Subsequently, Li et al. [25] proposes a task-oriented analysis method, which divides the data analysis work into two levels, focusing on the relevant visual AoI and the actions in the same task.

- Demonstrating or showcasing eye-tracking tools, systems, or frameworks specifically for data collection or visualization purposes:
  - Katona et al. [23] details the usage of Gazepoint GP3 eye-tracker and OGAMA software to record and analyze eye-motion parameters during a program source-code debugging taks. It emphasizes the recording process, calibration and data collection to understand how subjects solve errors in an algorithm.
  - Shaffer et al. [32], Sharif and Maletic [36], and Sharif et al. [35] introduce iTrace, an Eclipse plugin designed to record developers' eye movements while they work on software tasks.
  - Sharif et al. [34] provides practical guidelines in collecting and analyzing eye-tracking data through the iTrace software pipeline.
  - Fakhoury et al. [18] introduces iTrace-Atom and gazel, tools designed to enhance eye-tracking studies in software engineering by supporting source code edits. It addresses the problem of traditional eye-tracking tools struggling with tracking gaze data during source code editing, which limits the scope of research.
  - Behler et al. [14] introduces iTrace-Toolkit, a community eye-tracking infrastructure designed to standardize and simplify the analysis process, making it easier for researchers to handle large datasets and focus on relevant data for their studies.
  - Clark and Sharif [15] introduces iTraceVis, an eye-tracking visualization component for the iTrace plugin in Eclipse.
  - Mucke et al. [30] introduces REyeker, a remote eye-tracking tool.
  - Roy et al. [31] introduces VITALSE, a tool designed for visualizing combined eye-tracking and biometric data mapped to source code elements.
- Proposing an eye-tracking-based framework designed to enhance developer productivity:
  - Ahrens et al. [12] proposes an approach to help developers navigate and understand unfamiliar code more effectively. It uses eye-tracking to identifying areas of high and low attention during software maintenance tasks. Transfer attention by heatmaps in the code's background, and coloring the class name in the package explorer to represent class switches.

* Mangaroska et al. [29] presents a mirroring tool and analyzes how it influences users' performance in a debugging task; It applies multimodal user-centered analysis to design learning strategies that enhance programming and debugging skills.
* Deitelhoff and Harrer [16] shows a work-in-progress proposal to use eye-tracking data to create a dynamic help system for learners solving programming tasks.

.3 TABLE OF SELECTED STUDIES

Tabela 2 – Topic domains, eye-tracker, artifacts, and participants (S = Students, FM = Faculty members, P = Professionals, and NM = Not mentioned) for the selected papers.

Code	Topic Domain	Eye-trackers	Artifacts
Peterson et al., 2019a	Code Comprehension	Tobii X60	Java
Peterson et al., 2021	Code Comprehension	Tobii X60	Java
Peng et al., 2016	Debugging	Tobii X60	Java
Rodeghero et al., 2015	Code Comprehension	Tobii TX300	Java
Simhandl et al., 2019	Code Comprehension	Pupil Labs	CASM
Abid et al., 2019	Code Comprehension	Tobii X60	Java
Nivala et al., 2016	Debugging	SMI	C
Barik et al., 2017	Debugging	GazePoint GP3	Java
Andrzejewska and Stolińska, 2022	Model Comprehension	SMI iViewX	Pseudocode; Flowc
Busjahn et al., 2015	Code Comprehension	SMI REDm	Java
Beelders, 2022	Code Comprehension	NM	C#
Santos and Sant' Anna, 2019	Debugging	Tobii EyeX	C++
Jbara and Feitelson, 2015	Code Comprehension	EyeTribe	Java
Bauer et al., 2019	Code Comprehension	Tobii EyeX	Java
Peachock et al., 2017	Code Comprehension	Tobii X60	C++
Peterson, 2021	Code Comprehension	NM	Java; SQL
Störrle et al., 2018	Model Comprehension	SMI RED 4	UML
Chandrika et al., 2017	Debugging	SMI Red Professional	C#
Saddler et al., 2020	Code Comprehension	Tobii X60	Java
McChesney and Bond, 2021	Code Comprehension	Tobii X3-120	Java
Katona et al., 2019	Debugging	GazePoint GP3	C#
Kevic et al., 2015	Traceability	Tobii X60	Java
Lin et al., 2016	Debugging	Eyelink 1000	C
Katona et al., 2020	Code Comprehension	GazePoint GP3	C#
Melo et al., 2017	Debugging	Tobii EyeX	Java
Peterson et al., 2019b	Code Comprehension	Tobii X60	C++
Peitek et al., 2020	Code Comprehension	Tobii EyeX	Java
Gorski et al., 2022	Code Comprehension	NM	Javascript