Improving Industrial Cybersecurity Training: Insights into Code Reviews Using Eye-Tracking

Samuel Riegel Correia
Instituto Universitário de Lisboa (ISCTE-IUL), ISTA, Portugal

Maria Pinto-Albuquerque
Instituto Universitário de Lisboa (ISCTE-IUL), ISTAR, Portugal

Tiago Espinha Gasiba
Siemens AG, München, Germany

Andrei-Cristian Iosif
Universität der Bundeswehr München, Germany
Siemens AG, München, Germany

Abstract

In industrial cybersecurity, effective mitigation of vulnerabilities is crucial. This study investigates the importance of code reviews among cybersecurity professionals and analyses their performance in identifying vulnerabilities using eye-tracking technology. With the insights gained from this study, we aim to inform future tools and training in cybersecurity, particularly in the context of code reviews. Through a survey of industry experts, we reveal what tasks industry professionals consider the most important in mitigating cybersecurity vulnerabilities. A study was conducted to analyse how industrial cybersecurity professionals look at code during code reviews. We determined the types of issues our participants most easily discovered and linked our results with patterns and data obtained from an eye-tracking device used during the study. Our findings underscore the pivotal role of code reviews in cybersecurity and provide valuable insights for industrial professionals and researchers alike.

2012 ACM Subject Classification Security and privacy → Software and application security; Software and its engineering → Collaboration in software development; Information systems → Open source software; Security and privacy → Vulnerability management

Keywords and phrases code review, cybersecurity, development lifecycle, eye-tracking

Digital Object Identifier 10.4230/OASIcs.ICPEC.2024.17

Funding This work is partially financed by Portuguese national funds through FCT – Fundação para a Ciência e Tecnologia, I.P., under the projects FCT UIDB/04466/2020 and FCT UIDP/04466/2020. Furthermore, the first and second author thank the Instituto Universitário de Lisboa and ISTAR, for their support. Tiago Gasiba and Andrei-Christian Iosif acknowledge the funding provided by the Bundesministerium für Bildung und Forschung (BMBF) for the project CONTAIN with the number 13N16585.

1 Introduction

The security of software applications is crucial in today's digital landscape, in which cybersecurity threats continue to evolve in their sophistication and frequency [1]. As developers strive to create secure programs, analysing and understanding developers' cognitive processes when creating secure programs becomes crucial.

Code reviewing, a key task in the software development lifecycle, plays a pivotal role in creating secure code. It serves as a crucial measure in detecting vulnerabilities and other issues in code, making it an important task in the development of secure programs. The task of conducting a code review primarily involves reading code, this makes the use of eye-tracking technologies a fitting approach when studying developers' cognitive processes.

Although eye-tracking has been used in several coding and code interpretation studies, very few studies have focused on cybersecurity. This study aims to explore this underrepresented yet crucial subcategory of eye-tracking studies.

We have two main research questions:

RQ1 What tasks in the software development life cycle do industrial cybersecurity professionals consider to be the most crucial in mitigating cybersecurity vulnerabilities?

RQ2 How do industrial cybersecurity professionals analyse code while attempting to find cybersecurity vulnerabilities?

- **a** How successful are industrial cybersecurity professionals at finding cybersecurity vulnerabilities while performing code reviews?
- **b** Is there a relation between the patterns revealed using eye-tracking technology and the code reviewers' success in spotting the vulnerabilities?

By gaining insight into the thought processes of industrial cybersecurity professionals during code reviews, we seek to determine which practices are best suited to enhancing software security. With this information, we can suggest how to improve training, resources, and processes to enhance development practices and make them more secure.

2 Related Work

Related work in this field includes literature reviews of objectives and techniques important to analysing software developers' coding behaviour (e.g., [5]). Considering the current state of the art, we determined the following tasks to be crucial in cybersecurity and, additionally, be suitable candidates for analysis through an eye-tracking study:

a) Code reviewing
b) Analysis of code review tool outputs
c) Reading documentation
d) Researching online resources (e.g. Stack Overflow or other community-based resources)

From these, code reviews stood out as the most promising and interesting task to study. Code reviews are essential in detecting vulnerabilities and other issues in code. They are commonplace in development lifecycles and have been set as requirements in industrial standards such as ISO/IEC 62443.

Eye-tracking technology has been used in software engineering research to study various tasks in the development lifecycle, such as code comprehension, debugging, and code reviews [7]. This technology can record multiple aspects of participant behaviour, including visual focus, attention, interactions, and reading patterns, making it highly useful for research studies.

Generally, the articles related to eye-tracking use in cybersecurity are closely related to education in some form or another. These articles are either directly related to how we can create better pedagogical frameworks to educate individuals on cybersecurity (e.g., [4], [2]), or how we could adapt provided learning materials such as API documentation to further safe cybersecurity practices (e.g., [6]).

Most articles about programming are also relevant to cybersecurity, as analysing how individuals look at code also provides insight into how they deal with specific cybersecurity challenges.

3 Methodology

To answer RQ1, we developed a survey in which participants were asked to evaluate the four tasks mentioned previously: code reviewing, analysis of code review tool outputs, reading the documentation, and researching online resources, in terms of their perceived importance to cybersecurity using a five-point Likert scale. Besides obtaining the participants' opinions on the importance of these tasks, we also acquired some of their background information to help us describe the respondents. All participants are industrial cybersecurity professionals currently actively working in this field.

As for RQ2, we found that the best approach to answering this question lies in creating a study in which participants are presented with several code snippets and, for each one, are tasked with determining any vulnerabilities present in the code. The task we created was designed to simulate a code review.

The survey and the eye-tracking experiments were conducted with the same individuals, with the survey being conducted with participants before the code review experiment. These two parts, on average, took ≈ 8 and ≈ 21 minutes respectively. Our study was conducted in April 2024.

For this study, we used the Gazepoint GP3 Eye-tracking device in conjunction with the Open Gaze and Mouse Analyzer (OGAMA)[1] software which was used to create, record, and analyse the experiments.

Participants were presented with code snippets in C++ representing the five most common vulnerabilities according to the number of registered occurrences on CVEdetails.com[2]. These vulnerabilities' CWE IDs are CWE-79, CWE-119, CWE-89, CWE-20, and CWE-787. The code snippets were ordered in terms of difficulty, from the least to the most complex to analyse:

1. CWE-787 – Out-of-bounds Write
2. CWE-119 – Buffer Overflow
3. CWE-20 – Improper Input Validation
4. CWE-89 – SQL Injection
5. CWE-79 – Cross-site Scripting

4 Results

Participants

As mentioned previously, all participants in our study are industrial cybersecurity professionals currently working in this field. Besides answering RQ1, the survey we created allowed us to obtain some background information on the participants, which, in turn, would help us draw some conclusions from the phenomena we observed during the experiments.

A total of 12 individuals participated in the study. All were above the age of 24, with eight being between 25 and 34 years of age, and only one over 55. Two participants were female, and the remaining nine were male.

Regarding education, the participants have varying degrees, including one bachelor's degree, four doctorates, and seven master's degrees. Naturally, participants with higher degrees of education also had, generally, more years of work experience in cybersecurity. The participants with the least work experience in this field stated they had three years' worth of experience, and the most out of all participants was 25 years.

[1] http://www.ogama.net/
[2] https://www.cvedetails.com/

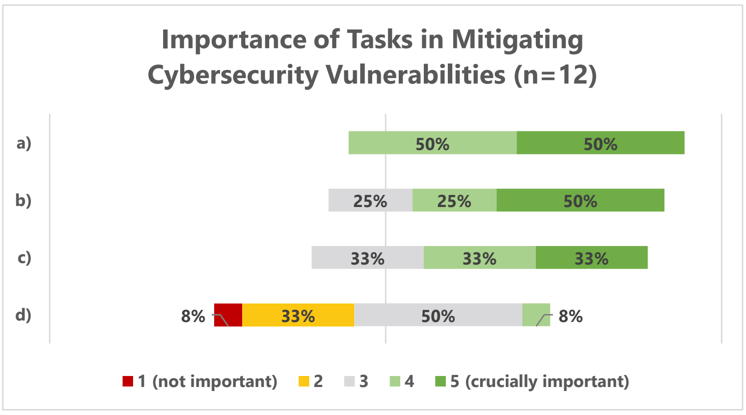


Figure 1 Importance given by participants to tasks in mitigating cybersecurity vulnerabilities.

RQ1 – What tasks in the software development life cycle do industrial cybersecurity professionals consider to be the most crucial in mitigating cybersecurity vulnerabilities?

Participants were asked to evaluate the following tasks on a scale from one (not important) to five (crucially important) regarding their assigned importance in mitigating cybersecurity vulnerabilities.

a) Code reviewing
b) Analysis of code review tool outputs
c) Reading documentation
d) Researching online resources (e.g. Stack Overflow or other community-based resources)

From this question, we obtained the results in Figure 1.

Professionals consider task a), code reviews, among the most critical tasks when mitigating cybersecurity vulnerabilities. In our survey, code reviewing was given an importance of four or five out of five by all of our survey participants, and many considered it the most important task.

Both b) and c) got similar results being considered, generally, less critical than a) but also having a quite positive average rating of ≈ 4.09 and ≈ 4.01 respectively.

One observation we made was that task d), which involves researching online resources, was considered relatively unimportant by our participants, with an average rating of approximately 2.4. Participants explained that community-based online resources are important for software developers when solving programming issues, but not ideal for addressing cybersecurity vulnerabilities.

An open-ended question was also included in the survey, asking the participants if there were additional tasks they considered important in mitigating cybersecurity vulnerabilities. Many chose to answer this question, with the most common answers including code testing, penetration testing, and secure coding training/workshops, all significant activities and tasks in cybersecurity.

RQ2 a) – How successful are industrial cybersecurity professionals at finding cybersecurity vulnerabilities while performing code reviews?

For our analysis, we created an AOI, referred to as the target, around the lines of code in these snippets which we considered to contain the main vulnerability to be discovered by the participants.

Table 1 Results on the analysis of code snippets with cybersecurity vulnerabilities.

Vulnerability	CWE-787	CWE-119	CWE-20	CWE-89	CWE-79
Discovery Rate	17%	50%	42%	100%	83%
Avg. Time Analysed (s)	234	289	259	214	242
Avg. Time to Find Vuln. (s)	205	168	137	91	189
Avg. Time Analysing Target (s)	51	100	51	23	36

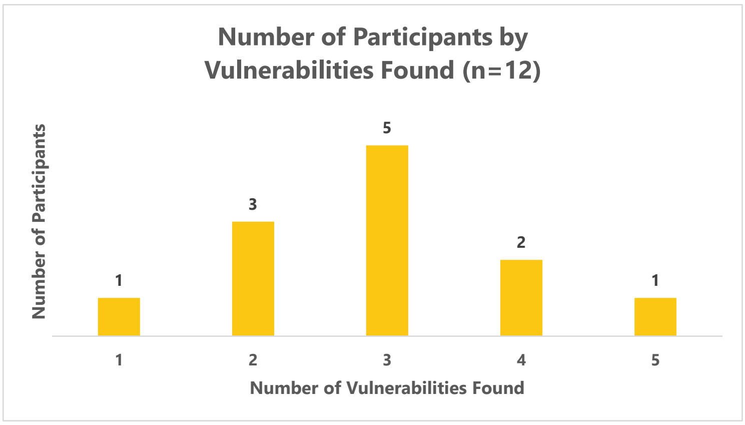


Figure 2 Number of Participants by Vulnerabilities Found.

The accuracy of responses varied considerably between the code snippets. For instance, in our first snippet of CWE-787, only two users identified the vulnerability in the code, while all users identified the vulnerability in the CWE-89 code snippet. Table 1 shows an overview of the results.

CWE-89 and CWE-79 stand out by being the most easily identified, by a considerable margin. These two vulnerabilities correspond to SQL injection and cross-site scripting (XSS), respectively, and are likely the most commonly discussed code vulnerabilities. This leads to the conclusion that the fact that these vulnerabilities are so well-known by professionals made them stand out and be easily identifiable.

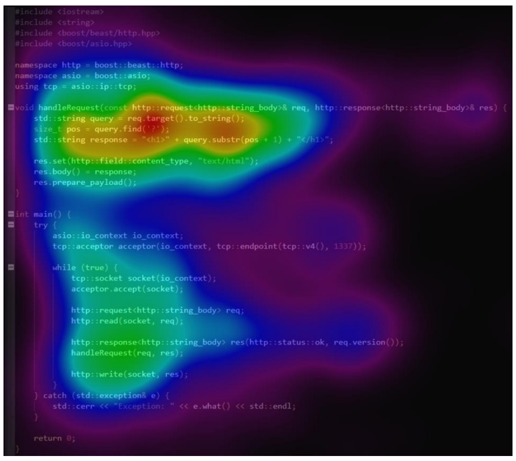
CWE-89 was the fastest to be found by our participants, while CWE-79 was one of the code snippets in which they took the longest to find vulnerabilities, even though these programs were quite similar in size. By watching the recordings made with the eye-tracking software, we see that our participants followed the code's execution path, which, for the code snippet on CWE-79, took fairly long before encountering the vulnerable code.

RQ2 b) – Is there a relation between the patterns revealed using eye-tracking technology and the code reviewers' success in spotting the vulnerabilities?

The average number of vulnerabilities found per participant is ≈ 3, Figure 2 shows the distribution of the participants by the number of vulnerabilities they detected out of the five considered.

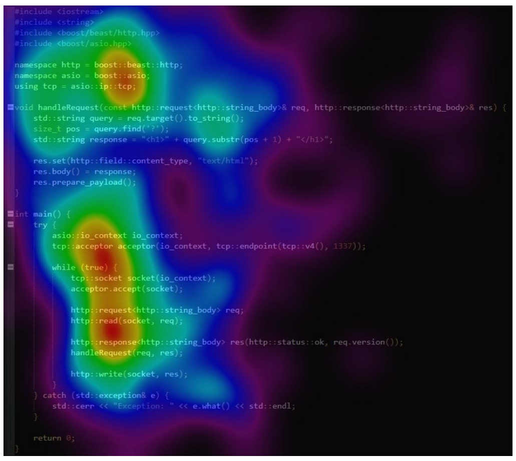
When comparing the number of vulnerabilities found with the background information provided by participants, we found no strong correlations. The years of experience and educational degree showed correlation values of about 0.08 and 0.31, respectively.

Next, we compared the results of participants who discovered the vulnerability, with those who did not, for each vulnerability. From this, we found that, on average, participants who didn't find the vulnerabilities looked at the code snippets and targets longer than those who



```
#include <iostream>
#include <string>
#include <boost/beast/http.hpp>
#include <boost/asio.hpp>

namespace http = boost::beast::http;
namespace asio = boost::asio;
using tcp = asio::ip::tcp;

void handleRequest(const http::request<http::string_body>& req, http::response<http::string_body>& res) {
    std::string query = req.target().to_string();
    size_t pos = query.find('?');
    std::string response = "<h1>" + query.substr(pos + 1) + "</h1>";

    res.set(http::field::content_type, "text/html");
    res.body() = response;
    res.prepare_payload();
}

int main() {
    try {
        asio::io_context io_context;
        tcp::acceptor acceptor(io_context, tcp::endpoint(tcp::v4(), 1337));

        while (true) {
            tcp::socket socket(io_context);
            acceptor.accept(socket);

            http::request<http::string_body> req;
            http::read(socket, req);

            http::response<http::string_body> res(http::status::ok, req.version());
            handleRequest(req, res);

            http::write(socket, res);
        }
    } catch (std::exception& e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

(a) CWE-89 Heatmap – Participants who discovered the vulnerability.



```
#include <iostream>
#include <string>
#include <boost/beast/http.hpp>
#include <boost/asio.hpp>

namespace http = boost::beast::http;
namespace asio = boost::asio;
using tcp = asio::ip::tcp;

void handleRequest(const http::request<http::string_body>& req, http::response<http::string_body>& res) {
    std::string query = req.target().to_string();
    size_t pos = query.find('?');
    std::string response = "<h1>" + query.substr(pos + 1) + "</h1>";

    res.set(http::field::content_type, "text/html");
    res.body() = response;
    res.prepare_payload();
}

int main() {
    try {
        asio::io_context io_context;
        tcp::acceptor acceptor(io_context, tcp::endpoint(tcp::v4(), 1337));

        while (true) {
            tcp::socket socket(io_context);
            acceptor.accept(socket);

            http::request<http::string_body> req;
            http::read(socket, req);

            http::response<http::string_body> res(http::status::ok, req.version());
            handleRequest(req, res);

            http::write(socket, res);
        }
    } catch (std::exception& e) {
        std::cerr << "Exception: " << e.what() << std::endl;
    }

    return 0;
}
```

(b) CWE-89 Heatmap – Participants who did not discover the vulnerability.

Figure 3 Heatmaps for Code Snippet of CWE-89.

Table 2 Average fixation rates of participants (fixations per second).

Average	Standard Deviation	Minimum	Maximum
3.86	1.43	1.73	5.96

did. We also found moderate correlations which indicate that participants who discovered more vulnerabilities looked at the code and targets for less time with values ≈ -0.4 and ≈ -0.5, respectively.

Heatmaps were created which helped us compare the fixation times on different parts of the code and targets, revealing potential differences between participants who discovered vulnerabilities and those who did not; an example of this can be seen in Figure 3.

Between the various heatmaps we compared, we noticed that, for most code snippets, participants who correctly identified vulnerabilities spent considerably more time looking at the parts of the programs with vulnerable code. Furthermore, in some snippets, such as the one seen in the images above, the high performers had a more focused approach to analysing the code, concentrating on a few key program elements.

We also examined our participants' gaze paths to determine if their code-reading strategies somehow impacted their performance. However, we are not able to draw any definitive conclusions on this since all types of participants followed some recognisable patterns, such as following the instructions from the main method and stepping into the functions that are called.

We then looked at the fixation rates of our participants, finding a meaningful correlation of ≈ -0.59 between the time spent analysing the snippets and their fixations per second or fixation rate. This correlation indicates that, on average, users with a higher fixation rate spent less time reading the code. We also found that the fixation rates were fairly consistent for each individual but varied quite significantly between the participants. Statistics on the fixation rates of our participants can be seen in Table 2.

When comparing the number of vulnerabilities found to each participant's average fixation rate, we see that a moderate correlation of ≈ -0.36 exists between the number of vulnerabilities discovered and the fixation rate of participants. This value indicates that participants who discovered more vulnerabilities had, on average, a lower average fixation rate.

5 Discussion

From the survey we created, our participants indicated that, out of the four tasks they were presented with, code reviewing was the most important in mitigating cybersecurity vulnerabilities. This demonstrates this task's significance in industrial cybersecurity and underscores the need for robust and effective code review practices.

Researching online resources was seen as the least important task from the list we presented. Participants indicated that they gave this task low importance despite being commonplace in program development because, specifically from the perspective of mitigating cybersecurity vulnerabilities, community resources can be unreliable and industry-followed resources such as standards and official documentation are preferable. Additionally, multiple participants referenced tasks such as penetration testing and secure coding training as being very important.

Our code review experiments revealed some interesting results. Firstly, SQL Injection and XSS vulnerabilities were detected at a much higher rate than the other vulnerabilities related to issues with memory allocation and buffer over/underflows. We believe, participants identified these issues quickly because the patterns for SQL Injection and XSS were very well-known. This indicates a need for increased awareness and training on other cybersecurity vulnerabilities, particularly those related to memory management and buffer handling, which may not be as widely recognised or understood.

As for the time it took participants to discover each of the vulnerabilities, the code snippets we selected are not ideal for this comparison, as several factors have to be considered to compare the vulnerability detection time, other than the vulnerabilities themselves. To determine what kinds of vulnerabilities take longer to be found, an experiment must be created which addresses the following two problems: first, different types of vulnerabilities require different program structures giving context to the vulnerable code which may vary in their ease of interpretation. Second, the participants' code scan path must be carefully considered as, ideally, the time it takes a user to read any code snippet before reaching the vulnerable code should be the same; this point has been discussed in other publications (e.g., [3]).

Through heatmaps, we determined that participants who correctly identified vulnerabilities had a seemingly more focused approach in looking at the code, usually focusing on the parts of the code snippets containing the vulnerabilities. This leads us to believe that individuals with more knowledge of the vulnerabilities are quicker to find them as they look at the program more efficiently. These results warrant further investigation as we can use the insight into how successful code reviews are conducted to teach people how to replicate this success.

The scan paths of our participants were also analysed; this was accomplished by reviewing the eye-tracking recordings of our participants. Some publications have found patterns in the code analysis process of experts when compared to that of novices[3], however, at this time we are not able to determine any specific strategies which led to better results during the experiment.

The eye-tracking data we obtained also included information on the fixation rates of our participants. Some authors have linked higher fixation rates with increased effort, interest, and exploration, while lower fixation rates may indicate a lower efficiency in tasks such as finding vulnerabilities in code [7][8]. In our experiment, participants who had spent less time looking at the code snippets were found to have higher fixation rates possibly indicating higher involvement in reading the code. However, we also found that participants who discovered fewer vulnerabilities had higher average fixation rates, which may indicate an increased effort in interpreting these code snippets, possibly leading to worse performance.

A limitation of our survey and study was the small sample size, this was mostly a product of the selection criteria we applied when choosing our participants. This restricts our confidence in the results and the analysis we can conduct with the data.

As for the future direction of this work, a more in-depth data analysis will be conducted, additional AOI will be created to enhance our analysis, and other code snippets, besides those analysed here, will be considered.

6 Conclusions

We conducted a survey and study on code reviews with industrial cybersecurity professionals. Our main objectives included determining how important these experts consider code reviews, determining how well they perform during code reviews, and analysing their performance with the help of eye-tracking technologies.

We conclude that industry professionals consider code reviews critical in mitigating cybersecurity vulnerabilities. Tasks such as the consultation of community-based resources (e.g., Stack Overflow), are considered less than ideal for cybersecurity as the information may be unreliable, and better sources, such as industry standards and official documentation, are more appropriate in this field.

By a considerable margin, SQL Injections and Cross-site Scripting (XSS) vulnerabilities were the most commonly detected vulnerabilities. This result can be explained by the fact that these two vulnerabilities are some of the most well-known and frequently discussed cybersecurity vulnerabilities.

When comparing the performances of those who discovered vulnerabilities and those who did not, we found that those who correctly identified vulnerabilities had a more focused approach to analysing the code and a slightly lower fixation rate, previously linked to lower efficiency in tasks such as code reviews. This insight into how successful code reviews are conducted warrants further investigation as it may help us teach people how to replicate this success.

In our future work, we plan to analyse data with different methods and explore other code snippets included in the study which weren't discussed here. This will allow us to provide further insights into how cybersecurity education should be adapted to improve performance during code reviews.

In summary, our study emphasises the crucial role of code reviews in cybersecurity. We also saw the importance of following certain code analysis patterns, and that exposure to different vulnerabilities is invaluable for code reviewers as commonly discussed issues were easily recognised.

References

1 Federal Cyber Security Authority. The state of it security in germany in 2023. *Federal Office for Information Security*, 2023.

2 Leon Bernard, Sagar Raina, Blair Taylor, and Siddharth Kaza. Minimizing cognitive load in cyber learning materials -– an eye tracking study. In *ACM Symposium on Eye Tracking Research and Applications*, volume PartF169257. Association for Computing Machinery, May 2021. doi:10.1145/3448018.3458617.

3 Teresa Busjahn, Simon, and James H. Paterson. Looking at the main method - an educator's perspective. In Otto Seppälä and Andrew Petersen, editors, *Koli Calling '21: 21st Koli Calling International Conference on Computing Education Research, Joensuu, Finland, November 18 - 21, 2021*. Association for Computing Machinery, November 2021. doi:10.1145/3488042.3488068.

4 Daniel Kyle Davis and Feng Zhu. Understanding and improving secure coding behavior with eye tracking methodologies. In J. Morris Chang, Dan Lo, and Eric Gamess, editors, *Proceedings of the 2020 ACM Southeast Conference, ACM SE '20, Tampa, FL, USA, April 2-4, 2020*, ACM SE '20, pages 107–114, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3374135.3385293.

5 Daniel Kyle Davis and Feng Zhu. Analysis of software developers' coding behavior: A survey of visualization analysis techniques using eye trackers. *Computers in Human Behavior Reports*, 7, August 2022. doi:10.1016/j.chbr.2022.100213.

6 Peter Leo Gorski, Sebastian Möller, Stephan Wiefling, and Luigi Lo Iacono. 'i just looked for the solution!'on integrating security-relevant information in non-security api documentation to support secure coding practices. *IEEE Transactions on Software Engineering*, 48:3467–3484, September 2022. doi:10.1109/TSE.2021.3094171.

7 Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. Toward an objective measure of developers' cognitive activities. *ACM Transactions on Software Engineering and Methodology*, 30, May 2021. doi:10.1145/3434643.

8 Zohreh Sharafi, Bonita Sharif, Yann Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering*, 25:3128–3174, September 2020. doi:10.1007/s10664-020-09829-4.