Using Gaze Data to Examine how Developers Perform Code Review

Lucas Zamprogno
The University of British Columbia
Vancouver, British Columbia
lucas.zamprogno@alumni.ubc.ca

ABSTRACT

Little work has been done to understand how developers perform a code reviews. Collaborative code review is a common and important part of modern software development, and developers can dedicate a considerable amount of time performing these reviews. Without knowing how developers actually perform these reviews, it is hard to create tools and systems to effectively support them in these complex and important tasks. We performed a prototype eye-tracking experiment to capture how a developer reads through code review patches on GitHub and gathered qualitative data about their experience performing these reviews. These data could be used to influence tools and procedures for code review and to improve developer productivity as they perform their code reviews.

CCS CONCEPTS

• **Human-centered computing** → *Human computer interaction (HCI)*; • **Software and its engineering** → *Software verification and validation*;

KEYWORDS

code review, eye-tracking, program comprehension

1 INTRODUCTION

Code review is a core part of software development in teams. While there have been studies on the process as a whole, little has been done to understand how individual developers perform a code review. Code reviews take time, and if done poorly can lead to bugs or increased development costs [13]. If we do not know how developers do code review or the challenges they face, it is hard to develop tools or systems to help them perform this task.

We used an eye-tracking system to record where developers look while performing a code review to better understand how they approach understanding a change. Additionally we observed developers doing their code review using a 'think aloud' protocol, and then follow up with an interview to get their experiences of the process. We found that individual developers have different patterns of reading code diffs, though starting with a more-or-less top to bottom reading seems common. Developers priorities were similar, focusing on code consistency and best practices over directly searching for bugs. They differed however when it came to what code gets focused on. Different developers had different types of changes they felt more or less comfortable skipping.

The contributions of this paper are:

(1) A prototype tool to dynamically record code review gaze data.
(2) A model of how developers perform code reviews based on their gazes and interactions.
(3) An empirical study measuring how developers perform code review based on real commits, and their feedback of the code review process.

2 RELATED WORK

This paper extends previous work in the areas of code review, interaction monitoring of developers, and the application of eye-tracking to code understanding research. Related work in each area will be reviewed and contrasted with the current paper.

2.1 Code Review

Previous work has examined the code review process and its benefits. Bacchelli and Bird researched the code review process in development teams and found that while desired, defect finding was not the primary outcome of code review. Instead, the primary benefits were more interpersonal, including expanding team knowledge and context, as well as finding new solutions to problems [2]. Beller et al. also investigated outcomes of modern code review in open source projects and found that most changes were focused on long term maintenance [4]. McIntosh et al. found a relationship between code review coverage and code quality, measured as a lower occurrence of post-release defects [13]. Rigby et al. highlighted the beneficial elements of the open source model of code review, citing the practice of frequent incremental changes and self-selected reviewers invested in the project [17]. Gousios, Pinzger, and Deursen investigated how the pull based development model functions. They find similar benefits of fast turnaround and increased engagement, and also highlight that pull requests are rarely rejected for technical reasons [8]. This work differs from the above in that it focuses on the process by which a single reviewer reasons about their review as opposed to investigating the code review process as a whole.

2.2 Interaction Monitoring

Past research has looked at summarizing how developers spend their time working, often through monitoring addons for various IDEs. Minelli et al. recorded usage data from the Pharo IDE to break down how developers spend their time

when using the IDE [14]. Murphy, Kersten, and Findlater recorded how developers used various tools provided by the Eclipse IDE using the Mylar plugin [15]. Amann et al. also studied frequency of feature use within the Visual Studio IDE, along with how much time developers spent on various development tasks [1]. The primary shortcoming of these studies is the limitation of monitoring to interactions within an IDE. In contrast, we extend this monitoring to the web browser and apply it to the code review process, as well as including biometrics in the interaction monitoring.

2.3 Code Comprehension with Eye-tracking

Sharif, Falcone, and Maletic in partial replication of Uwano et al. tracked eye movements on a small code snippet while looking for defects. They found that those who scanned longer at first found defects quicker [18, 19]. This study shares many similarities to these previous efforts, but differs primarily on the content and presentation of the material under review. The Sharif and Uwano studies presented source code snippets 12 to 23 lines long in a custom viewing application, while our study works with any content found on GitHub. Kevic et al. captured interaction and gaze data during a software change task on a more complex system. They found that gaze data was more rich than just IDE interaction data, and demonstrated patterns and differences of professional and student developers navigating a code snippet [11]. Fritz et al. used pupil dilation, saccades, and fixations, along with other physiological measures to to assess difficulty of code comprehension. Readings from these measures were able to be used to make predictions about task difficulty [7].

Multiple studies have looked at the general patterns followed when reading code, finding that it is different from normal prose, and becomes less linear the more experienced the programmer reading the code is [3, 5, 6]. Lin et al. found that effective students followed a program logically as opposed to linearly or narrowing in on a certain element [12]. Jbara and Feitelson reported that in code with a repetitive structure gaze times are longer on the initial repeated segments, then shorten after the pattern becomes familiar [10].

Nguyen, Vrzakova, and Bednarik developed a web tracking plugin that automatically annotates web elements for an eye tracking device [16]. Our plugin differs from this by capturing data from pre-defined but automatically identified elements, gathering context relevant data from this smaller subset of elements.

3 EXPERIMENTAL DESIGN

This section first presents the procedure used when participants performed their review, followed by details about the implementation of the eye tracking and data gathering system.

3.1 Method

The study participants were three graduate students and a professional programmer from the department of Computer Science at UBC. Study participants used a computer with the Tobii Eye Tracker attached, and performed the default eye tracker calibration process. Participants then logged into their account on GitHub and navigated to a pull request they were going to review. Two participants reviewed pull requests from software systems they worked on in the past, while the other reviewed a more recent one that they had not looked at in depth. Participants were instructed to conduct their review as they normally would, while talking aloud about their process. While they conducted the review, the researchers made their own observations about how the subject was approaching their review. Whenever the participant considered themself finished, the tracker was stopped and the study moved to the interview phase. Participants were asked a series of questions related to general code review procedures, any observations they had about what they looked for while performing this review, and their thoughts about code review generally.

3.2 Support

This prototype study relied on the creation of tools to interact with the eye-tracker, apply the gaze data to get information relevant the code review, and present it in a more intuitive format. This section describes at a high level how these tools function.

3.2.1 Application. The eye tracking device used for the study was a Tobii Eye Tracker 4C which has a 90Hz refresh rate, however no official precision or accuracy values have been provided. A small C++ application sends gaze data from the tracker to a Google Chrome extension. Gaze data was passed filtered with Tobii's 'lightly filtered' preset, weighting data points based on age and velocity of eye movements. The extension uses coordinates from the eye tracker to identify the currently viewed element in the browser using the DOM `elementFromPoint` method. We then check if this element is, or is a child of, a predefined list of target elements as identified by jQuery selectors. If it is, once the gaze leaves this target element a gaze event is recorded along with details contextual on the element being looked at.

3.2.2 Contextual Data Gathering. The Chrome extension gathers data from the users gaze, mouse and keyboard input, and about the pages participants visited. All interaction data includes a hash of the page URL, a label for the page purpose (e.g. a commit or issue), timestamps, and a classification of the element being interacted with. Certain elements also record additional information, most importantly are the lines in a diff. An event for a diff line stores whether it was an addition, deletion or unchanged, its index, length, and level of indentation. What a participant looked at in a diff would not be very meaningful without a baseline, so the extension also gathers similar line data for every line of any diff found on the page. The most notable type of information not recorded was any semantic details of code; whether a line was a method header, statement, or comment for instance was not recorded.

Table 1: Subject 1 gaze statistics; This shows disinclination to additions and preference more indented lines. Asterisks indicates differences of 20% or more.

Line detail metric	In commit	Viewed
Additions, % of lines	65.7	43.9*
Additions, % of text	63.0	43.9*
Deletions, % of lines	15.0	23.4*
Deletions, % of text	15.7	23.4*
Unchanged, % of lines	19.2	32.7*
Unchanged, % of text	21.3	32.7*
Average indentation, spaces	4.85	6.76*
Average length, characters	14.2	16.9

Table 2: Subject 2 gaze statistics; This shows lessended attention to deletions. Asterisks indicates differences of 20% or more.

Line detail metric	In commit	Viewed
Additions, % of lines	56.4	61.7
Additions, % of text	62.2	61.7
Deletions, % of lines	33.2	26.8*
Deletions, % of text	33.9	26.8*
Unchanged, % of lines	10.4	11.5
Unchanged, % of text	4.0	11.5*
Average indentation, spaces	3.1	2.78
Average length, characters	22.8	23.3

3.3 Analysis

Baseline data to contrast what participants viewed was calculated using the diff data gathered. The proportion of each change type (addition, deletion, or unchanged) was calculated in two ways, by number of lines and by length. The number of lines better represents how much of the diff's actual space is taken by that change type, but length is more representative of how long it would take to read all of that change type. Other baseline data calculated was the average indentation, a possible proxy for complexity, and line length.

Gaze data was graphed as a scatterplot using using the Python `matplotlib` library, sampling the user's gaze location every 10 ms. To reduce noise, any gaze on an item less than 100 ms was removed[1]. The X-axis represents the point in time of the gaze, with the Y-axis as the index of the change relative to the full commit. Data about the original diff is used to scale the graph and show the change types associated with each line, with diffs separated by a black line. If the diff was viewed in split view where changes are side by side in the same line, further processing had to be done. To keep the data in two dimension but maintain change proximity, changes are reorganized to be adjacent vertically instead of horizontally.

4 RESULTS

This section presents the results from the prototype experiment, as well as comments and observations made during the review and subsequent interview. This study examines three questions:

(1) Can a tool be developed to automatically record and annotate gaze data on websites?
(2) How do developers perform code review?
(3) What do developers use when performing code reviews?

4.1 Gaze Results

Where devleopers focused during review is presented as a visual history, and through the properties of the lines they spent more time on.

[1]This value was chosen for removing most outlying points, while still being much shorter than fixations in a simple visual search [9].

4.1.1 Gaze Statistics. A sample of the statistics gathered from a review from Subject 2 is shown in Table 2. Statistics gathered from the reviews can be seen in Tables 1 and 2. The proportion of additions, deletions, and unchanged lines are compared to the proportion of time the reviewer spent viewing that type of change. Additionally the average indentation and length of lines in the commit are compared to the average indentation and length looked at during the review. Subject 1 spent less time on additions than other change types[2] , and more time on further indented lines. For Subject 2, time spent on each change type is roughly proportional to its representation in the commit, with a slight preference to additions and unchanged lines[3] over deletions. Similarly, indentation and line length closely resembled the commit itself. This would indicate that in this review, lines were not given notably more or less view time based on their specific properties. The duration of the gaze would correspond to the length of the points on the x-axis.

4.1.2 Gaze Histories. The gaze data is presented as a scatter-plot; the location of the gaze at a current time was sampled at even intervals. A horizontal line of points on the graphs would indicate the developer was gazing at the same line(s) of code for a period of time. Likewise a steady slope would indicate the developer is reading the code line by line, but not stopping at any particular point. A sharp vertical slope indicates very fast skimming or scrolling of the page. Figures 1 and 2 show gaze graphs resulting from code review sessions from two different participants. There are some similarities between the two charts. Both participants trend downward from the beginning of the review before later returning their gaze to code that has already been viewed. Additionally, both graphs show moments of focus on a small section of code. Subject 1 (Figure 1) can be seen to focus on two changed lines at the 60 second mark (roughly line 78), and Subject 2 (Figure 2) focuses on neighboring sets of lines

[2]The subject identified that the large block of additions appeared to be auto-generated as a result of a change made elsewhere in the system, and deemed it unimportant to review.
[3]In cases where a commit contains many small changes, unchanged line view time may be erroneously inflated as a gaze on a single changed line may be reported as being on a neighboring unchanged line instead due to tracker accuracy limitations.

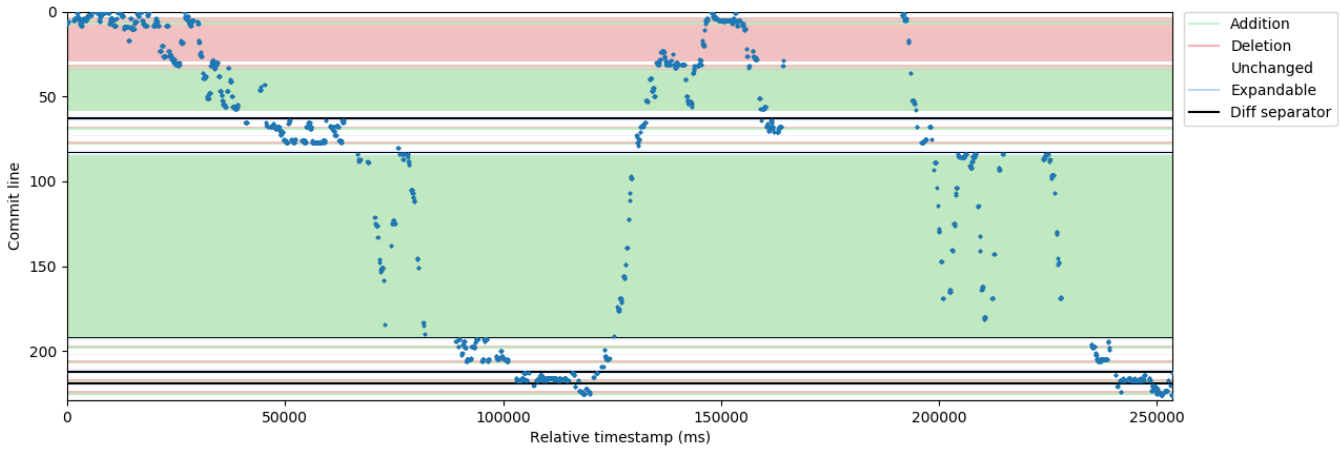

Figure 1: Subject 1 code review gaze record. Horizontal spans of dots show focus on certain areas early on. A large gap can be seen in the latter half, indicating a period when the developer was looking outside the diff.

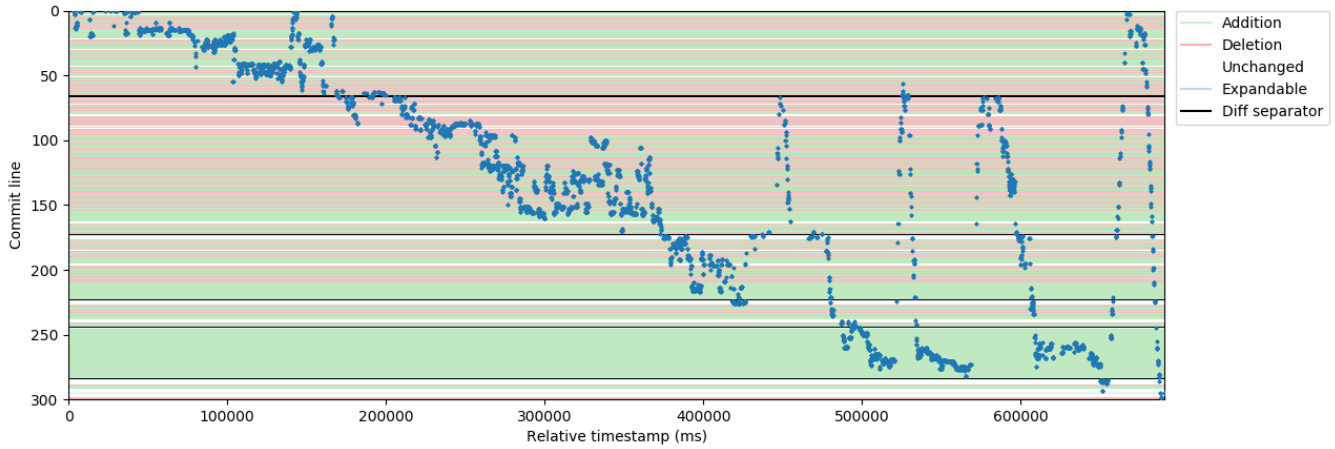

Figure 2: Subject 2 code review gaze record. A slow downward trend shows the code was read methodically from top to bottom, followed by a double-checking behaviour before viewing the final block. Many additions and deletions are interlaced as a result of being converted from a split view.

at the 2.5 minute mark (roughly line 45). Notably, the graphs have different structures overall: Subject 2 has a clear and consistently paced scan down the page, followed by multiple brief revisits across the commit. Subject 1 also has an initial downward scan but with more focusing and revisiting of lines, and later spends proportionally much longer jumping around, both between diffs and within each diff. There is also a clear break in the gaze data in the second half of Subject 1's review, during which the participant was looking for information outside the diff on a different page. Subject 2's graph shows no such gap, indicating the participant remained within the page for the duration of the review.

4.2 Interview results

During our interviews, no participants brought up defects or bugs as a main purpose of their review. Participants did however report trusting the author to have done the diligence of ensuring functional correctness of the code, provided they were familiar with the author (S1, S3). Participants reported that they looked at comments, titles, or related issues to get a better understanding (or to remember) the context for the change (S1, S3, S4). Most participants agreed that having IDE-style code navigation tools when looking at a diff would be helpful (S1, S2, S4), though one participant expressed a desire to keep the interface simple instead (S3). Some areas

showed more unique preferences between participants. One participant noted spending less time on large block changes (e.g. a large delete) because they believed it was less likely to contain mistakes or inconsistencies (S4), where another spent more time on those as they believed it would be more relevant to understanding the change (S2).

5 DISCUSSION

This section will discuss the limitations and threats to validity of the results, and outline suggestions for future work.

5.1 Limitations

While the Tobii Eye Tracker 4C is useful for its low cost and convenient deployment, its accuracy limits precision about what specific lines of code are being viewed. This, combined with the desire to be language independent, led to the decision to not look any semantics. For instance a line could be a method header, a control flow statement, or a comment, and they would be treated and recorded the same. In this study we wanted participants to be able to use either diff view, however when graphed split diffs had to have lines be re-ordered to keep them two dimensional. Our method of ordering the lines preserved the proximity of related lines, at the expense of clarity on whether the addition or deletion is being viewed. The view graphs ignore expanded lines or diffs, though this was not an issue in practice as none of the participants expanded collapsed lines or diffs.

5.2 Threats to Validity

Participants were not representative of a typical software development workplace. Most repositories were small personal or open source projects. The low sample size is sufficient as proof of concept, however not enough to draw solid conclusions from. During this study, the researchers were present for the time of the code review. While beneficial for hearing the developer talk aloud and interview the developer while the review is recent, being observed may cause developers to change their behaviour, most likely being more thorough than typical.

5.3 Future Work

A future improvement to the study could include a higher accuracy tracker and language parsing to introduce details about the code itself into the analysis. For instance, the ability to identify method headers or highlight control or data flow in the diff could lead to a better understanding of why developers view diffs the way they do. Another improvement could come from the removal of researches from the environment. A modification of the study could consider setting up the eye tracker and software at a developers workstation to run for an extended period of time. This could provide more natural behaviour in regards to time spent and gaze patterns. Handling of split diffs leads to an awkward representation if the gaze history needs to be kept two-dimensional. Future studies should consider restricting participants to only a unified diff view.

6 CONCLUSIONS

We conducted a prototype study to investigate how developers perform code review, focusing on the location and context of their visual focus. To do this, we built a tool to track gaze behaviour on GitHub using a commercial eye-tracker. Results showed that participants did an initial top to bottom read of a change without much revisiting of previous lines, though there was variance in thoroughness and amount of later revisiting. Additionally, overall focus did not seem to discriminate between lines of code based on their content. We hope this experiment can be expanded in future work to include a larger sample size of industrial developers working in the field to better understand how code review is performed so we can understand how to improve both code review tools and and code review processes to maximize the utility of time spent performing these important tasks.

A INTERVIEW QUESTIONS

The following are possible questions used during the interview. Questions may be skipped if not applicable or already answered during a response to a previous question.

- In general, how do you perform a code review?
- Do you or your company have a checklist or procedure for code reviews?
- Did you perform any of these reviews differently from a typical review? If so, how?
- What did you look for during the reviews?
- How familiar were you with the code you reviewed?
- What aspects of the code changes did you feel comfortable skipping? Why?
- What aspects of the code changes did you examine more closely? Why?
- Why did you seek information outside the unix diff?
- Did you think about how the change would impact code outside of the changed lines?
- What type of information do you hope to get from reading discussion comments?
- Why did you seek information outside GitHub?
- Was there information that you felt was missing or hard to find?
- Is there anything that stood out to you during a review that you decided not to document? Why?
- Are there any non-collaborative features that could help you better review the code?
- Are there any collaborative features that could help you better review the code?
- Is there any additional information that could help you better review the code?

ACKNOWLEDGMENTS

I acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), which invests annually over $1 billion in people, discovery and innovation. I would also like to thank all participants who volunteered to participate.

REFERENCES

[1] S. Amann, S. Proksch, S. Nadi, and M. Mezini. 2016. A Study of Visual Studio Usage in Practice. In *International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 124–134. https://doi.org/10.1109/SANER.2016.39

[2] A. Bacchelli and C. Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *International Conference on Software Engineering (ICSE)*. 712–721. https://doi.org/10.1109/ICSE.2013.6606617

[3] Roman Bednarik and Markku Tukiainen. 2006. An Eye-tracking Methodology for Characterizing Program Comprehension Processes. In *Proceedings of the Symposium on Eye Tracking Research & Applications (ETRA)*. 125–132. https://doi.org/10.1145/1117309.1117356

[4] Moritz Beller, Alberto Bacchelli, Andy Zaidman, and Elmar Juergens. 2014. Modern Code Reviews in Open-source Projects: Which Problems Do They Fix?. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. 202–211. https://doi.org/10.1145/2597073.2597082

[5] Teresa Busjahn, Roman Bednarik, Andrew Begel, Martha Crosby, James H. Paterson, Carsten Schulte, Bonita Sharif, and Sascha Tamm. 2015. Eye Movements in Code Reading: Relaxing the Linear Order. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. 255–265. http://dl.acm.org/citation.cfm?id=2820282.2820320

[6] Martha E. Crosby and Jan Stelovsky. 1990. How Do We Read Algorithms? A Case Study. *Computer* 23, 1 (Jan. 1990), 24–35. https://doi.org/10.1109/2.48797

[7] Thomas Fritz, Andrew Begel, Sebastian C. Müller, Serap Yigit-Elliott, and Manuela Züger. 2014. Using Psycho-physiological Measures to Assess Task Difficulty in Software Development. In *Proceedings of the International Conference on Software Engineering (ICSE 2014)*. 402–413. https://doi.org/10.1145/2568225.2568266

[8] Georgios Gousios, Martin Pinzger, and Arie van Deursen. 2014. An Exploratory Study of the Pull-based Software Development Model. In *Proceedings of the International Conference on Software Engineering (ICSE)*. 345–355. https://doi.org/10.1145/2568225.2568260

[9] Harold H Greene. 2006. The Control of Fixation Duration in Visual Search. *Perception* 35, 3 (2006), 303–315. https://doi.org/10.1068/p5329 PMID: 16619948.

[10] Ahmad Jbara and Dror G. Feitelson. 2015. How Programmers Read Regular Code: A Controlled Experiment Using Eye Tracking. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. 244–254. https://doi.org/10.1007/s10664-016-9477-x

[11] Katja Kevic, Braden M. Walters, Timothy R. Shaffer, Bonita Sharif, David C. Shepherd, and Thomas Fritz. 2015. Tracing Software Developers' Eyes and Interactions for Change Tasks. In *Proceedings of the Joint Meeting on Foundations of Software Engineering (ESEC/FSE)*. 202–213. https://doi.org/10.1145/2786805.2786864

[12] 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. *IEEE Transactions on Education* 59, 3 (Aug 2016), 175–186. https://doi.org/10.1109/TE.2015.2487341

[13] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. 2014. The Impact of Code Review Coverage and Code Review Participation on Software Quality: A Case Study of the Qt, VTK, and ITK Projects. In *Proceedings of the Working Conference on Mining Software Repositories (MSR)*. 192–201. https://doi.org/10.1145/2597073.2597076

[14] Roberto Minelli, Andrea Mocci and, and Michele Lanza. 2015. I Know What You Did Last Summer: An Investigation of How Developers Spend Their Time. In *Proceedings of the International Conference on Program Comprehension (ICPC)*. 25–35. https://doi.org/10.1109/ICPC.2015.12

[15] Gail C. Murphy, Mik Kersten, and Leah Findlater. 2006. How Are Java Software Developers Using the Eclipse IDE? *IEEE Softw.* 23, 4 (July 2006), 76–83. https://doi.org/10.1109/MS.2006.105

[16] Duc Nguyen, Hana Vrzakova, and Roman Bednarik. 2016. WTP: Web-tracking Plugin for Real-time Automatic AOI Annotations. In *Proceedings of the International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct (UbiComp)*. 1696–1705. https://doi.org/10.1145/2968219.2968338

[17] P. Rigby, B. Cleary, F. Painchaud, M. A. Storey, and D. German. 2012. Contemporary Peer Review in Action: Lessons from Open Source Development. *IEEE Software* 29, 6 (Nov 2012), 56–61. https://doi.org/10.1109/MS.2012.24

[18] Bonita Sharif, Michael Falcone, and Jonathan I. Maletic. 2012. An Eye-tracking Study on the Role of Scan Time in Finding Source Code Defects. In *Proceedings of the Symposium on Eye Tracking Research and Applications (ETRA)*. 381–384. https://doi.org/10.1145/2168556.2168642

[19] Hidetake Uwano, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. 2006. Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. In *Proceedings of the Symposium on Eye Tracking Research &Amp; Applications (ETRA)*. 133–140. https://doi.org/10.1145/1117309.1117357