GANDER: a Platform for Exploration of Gaze-driven Assistance in Code Review

Saranpää, William; Apell Skjutar, Felix; Heander, Johan; Söderberg, Emma; Niehorster, Diederick C; Mattsson, Olivia; Klintskog, Hedda; Church, Luke

2023

Document Version:
Publisher's PDF, also known as Version of record

GANDER: a Platform for Exploration of Gaze-driven Assistance in Code Review

William Saranpää
william.saranpaa@cs.lth.se
Lund University
Sweden

Felix Apell Skjutar
felix.apellskjutar@gmail.com
Lund University
Sweden

Johan Heander
johan.heander@cs.lth.se
Lund University
Sweden

Emma Söderberg
emma.soderberg@cs.lth.se
Lund University
Sweden

Diederick C. Niehorster
diederick_c.niehorster@humlab.lu.se
Lund University
Sweden

Olivia Mattsson
oliviamattssons@gmail.com
Knowit
Sweden

Hedda Klintskog
hklintskog@gmail.com
Unibap
Sweden

Luke Church
luke@church.name
Lund University
Sweden

ABSTRACT

Gaze-control and gaze-assistance in software development tools have so far been explored in the setting of code editing, but other developer activities like code review could also benefit from this kind of tool support. In this paper, we present GANDER, a platform for user studies on gaze-assisted code review. As a proof of concept, we extend the platform with an assistant that highlights name relationships in the code under review based on gaze behavior, and we perform a user study with 7 participants. While the participants experience the interaction as overwhelming and lacking explicit actions (seen in other similar user studies), the study demonstrates the platform's capability for mobility, real-time gaze interaction, data logging, replay and analysis.

CCS CONCEPTS

• **Human-centered computing** → **HCI design and evaluation methods**; **Empirical studies in HCI**; • **Software and its engineering** → **Software verification and validation**.

KEYWORDS

eye-tracking, code review, human-computer interaction, gaze data

ACM Reference Format:
William Saranpää, Felix Apell Skjutar, Johan Heander, Emma Söderberg, Diederick C. Niehorster, Olivia Mattsson, Hedda Klintskog, and Luke Church. 2023. GANDER: a Platform for Exploration of Gaze-driven Assistance in Code Review. In *2023 Symposium on Eye Tracking Research and Applications (ETRA '23), May 30–June 02, 2023, Tubingen, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3588015.3589191

ETRA '23, May 30–June 02, 2023, Tubingen, Germany

ACM ISBN 979-8-4007-0150-4/23/05.
https://doi.org/10.1145/3588015.3589191

1 INTRODUCTION & RELATED WORK

Tool-based code review has become an established practice in industry [Bacchelli and Bird 2013; Sadowski et al. 2018]; it provides benefits like quality assurance and knowledge sharing, but it typically also takes a lot of developer time. The provided code review tools, which typically is a shared central web service centered around a textual diff view (e.g., GitHub[1], Gerrit[2], Critique [Sadowski et al. 2018]), may also not align well with developers' needs; causing them to, for instance, spend additional time to move code to their IDE for review [Söderberg et al. 2022]. There is a need for better code review tooling, and with developers spending a lot of time looking at code trying to understand it, there is an opportunity to approach such tool improvements with the assistance of gaze data [Kuang et al. 2023].

The use of eye-tracking in user studies and experiments focused on software development activities has increased in recent years [Kuang et al. 2023; Obaidellah et al. 2018; Sharafi et al. 2020, 2015], including studies of developers' behavior in code review (e.g., [Begel and Vrzakova 2018; Vrzakova et al. 2020]). There has been work on accelerating research using gaze data in code editing with iTrace [Guarnera et al. 2018], which plugs into your development environment to gather gaze data and assists with post-processing of it. For real-time processing of gaze data, the focus so far has been on code editing: EyeDE [Glücker et al. 2014] explores gaze-based control and assistance in code editing; EyeNav [Radevski et al. 2016] explores gaze control with focus on navigation in code with scrolling, moving of the cursor, and selection; CodeGazer [Shakil et al. 2019] explores pure gaze-based interaction to trigger code navigation support in editing; and Javardeye [Santos 2021] explores gaze control in code editing focused on positioning of the cursor. We are not aware of any existing approaches that explore gaze-driven control or assistance in code review.

In this paper, we investigate how to explore gaze-assistance in code review by focusing on the setup needed for such exploration. To this end, we present a design and implementation of a

[1]https://github.com/
[2]https://www.gerritcodereview.com/

platform[3] for exploration of gaze-assistance in code review (Section 2), then we present how we implemented the first version of a gaze-assistant for highlighting of name relationships (Section 3). We further describe an exploratory early evaluation of the gaze-assistant (Section 4). We then discuss our results in relation to the related work (Section 5) and conclude with ideas for future work (Section 6).

2 THE GANDER PLATFORM

We set out to design and implement a platform for exploration of gaze-assistance in code review, with the goals of enabling 1) realistic data gathering, 2) low-cost exploration of gaze-assistance, and 3) a fully functioning experimental setup for data gathering. For realistic data gathering, we sought a design that would resemble the code review interface found in popular version control tools, like GitLab[4] and GitHub[5], used in software development. We further wanted a mobile setup that would allow for data gathering in a non-lab setting closer to practitioners. For low-cost exploration of gaze-assistance, we sought a design that would allow for exploration of the user interaction layer without too much effort. For a fully functioning experimental setup for data gathering, we sought a design that would be robust enough to reliably gather data in an experimental setup. We further wanted a design that would be capable of providing sufficient performance for real-time analysis of gaze data.

2.1 System walk-through: from interaction to reaction

When entering the frontend, the user is greeted with a login form and a button to toggle on the eye tracker. Upon login, the user can either replay a previous session or view a list of pull requests connected to a certain repository. The user can choose a pull request from this list and is then shown the code changes that are part of the pull request, see Fig. 1 (left). When a user enters the page of a specific pull request, all relevant pull request information is fetched from GitHub. The comments for each file from GitHub are displayed and the user can respond to or create their own comments as well.

Interactions (e.g., mouse movement, button clicks, scrolling) and gaze data are continuously logged during a session, along with data received from GitHub. The fixation points and interactions are stored as JSON and the raw eye-tracking data is stored as TSV. This data enables offline replay of user sessions, either for further analysis by a researcher, or for cued retrospective recall for the user in an experiment. In the latter use case, an estimate of the thoughts of a participant can be gathered while not interrupting the gaze data by letting them think aloud [Van Gog et al. 2005]. The replay logs are device-independent and enable running replays on different machines than that used during the session, although it would have to be the same browser and screen resolution. The logs can also be used for offline analysis using any general data analysis framework such as SciPy, R, Octave, etc., in which case screen resolution or browser does not matter.

[3]Open source at https://gitlab.com/lund-university/gander
[4]https://about.gitlab.com/
[5]https://github.com/

To replay a session of a participant, the user selects replay mode and chooses which of the locally saved sessions to replay. The view is then overlaid with a violet hue and an exit button is added in the upper left corner to be able to exit replay mode, see Fig. 1 (right). In replay mode, the user's mouse movements are shown by means of a drawn cursor, and their gaze position is visualized by means of a circle. The user can pause the replay and interact with the view, but can only enter into files part of the interaction recorded in the log during their session.

2.2 Architecture

The platform architecture (illustrated in Fig. 2) is structured around several microservices as the backend and a web application as the frontend. Communication between services and the frontend is done using REST APIs and websockets.

The **GANDER client** is written in Dart and acts as the frontend of the application and creates the views as well as processing data related to replay. The client logs clicks, mouse movements, keyboard input and scrolling. Scroll position is saved and used during replay to place the gaze data in the correct place in the view. To detect fixation points in real-time, we used the finite state machine of Diaz-Tula et al. [Diaz-Tula and Morimoto 2017] implemented as a microservice in python. Gaze data is first pulled from the eye tracker using Titta [Niehorster et al. 2020] and streamed to the microservice using websockets. Classified fixation points are then streamed to the GANDER client over another websocket interface.

The **GANDER service** is a broker service responsible for connecting the different microservices with the goal of simplifying the communication between the frontend and the backend. The GANDER service as well as the rest of the microservices are written in java. The microservices include the **GitHub connector**, which fetches information using the GitHub API by gathering pull requests in a repository. For each pull request, modified files and comments are retrieved and stored locally. The GitHub connector service then acts as the code repository, caching the file contents for each pull request while the comments are sent to the annotation service.

The **annotation service** is responsible for managing all the comments within a pull request. It fetches the existing ones from the GitHub connector and also stores the ones written by a user in the frontend. This allows a user to answer in an existing thread of comments and also to start a new comment thread. By storing the user interactions, the comments can be recreated from the logs of a session. Comments generated during a user session are not pushed to the official GitHub repository, they are only stored locally.

The **diff service** provides information about changes between different versions of a file so that the frontend can display the diff view. It uses the GitHub API to get the lines that have been changed between versions of a file. When the content of a line has been changed, and not added or deleted, the diff service compares the versions of the line to find out which parts have been changed. If only a small part of the line has been changed, the coordinates of the changes are communicated so that interline-changes can be visualized with small local highlights.

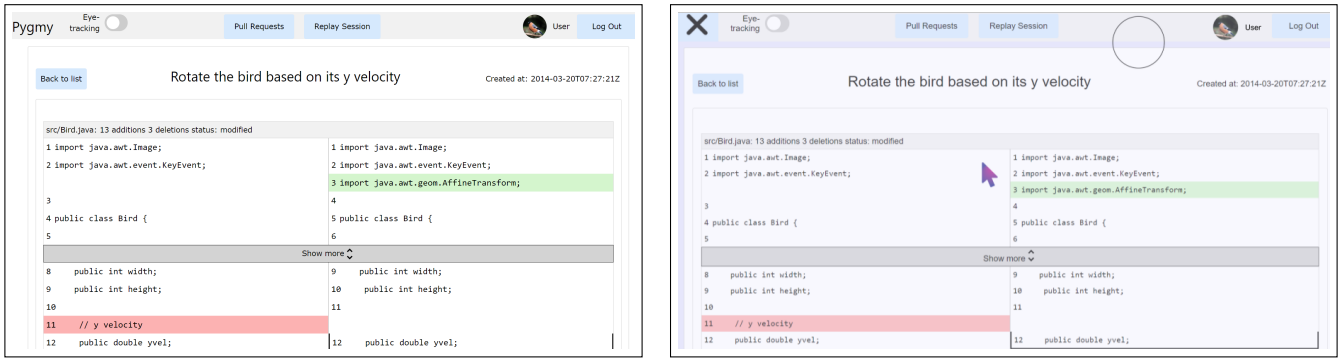

Figure 1: Different types of views for the user: interactive session (left) and replay session (right).

When a user opens a pull request, its code is sent to the **compiler service**, which compiles the file using the ExtendJ Java compiler [Ekman and Hedin 2007], thus requiring repositories written in Java. The compiler service determines the declaration and uses of variables, fields, methods and parameters for each file, and makes this information available to the frontend. The **storage service** stores the replay logs containing each interaction with the frontend, the eye tracking data and requests and responses to and from the REST APIs. The **replay service** uses the logs stored by the storage service to replay a previous session for user review or qualitative analysis. We separate the information into a work list, consisting of gaze data and interaction with the frontend, as well as a request and response map to mock the backend. This means that the eye movement, interaction and state of the repository remain the same as when the user did the session.

3 GAZE-ASSISTED NAME RELATIONSHIP HIGHLIGHTING: THE RAIN CLOUD STRATEGY

To show the capabilities of the platform, as a proof of concept we decided to implement a gaze-assisted highlighting of the relationship between the declaration and uses of variables, methods, fields, etc., see Fig. 3. We call this highlighting approach "the rain cloud strategy" since it resembles a rain cloud at the declaration with traces falling down like rain to every use.

Here, we let the names of variables, methods etc. be areas of interest (AOIs) and highlight those and the relationships between them, for instance, between the declaration of a variable and the uses of the variable. When a minimum number of fixation points are located within an AOI, it is faintly highlighted. As the AOI receives more fixations, the highlighting of the AOI intensifies in color and size and connected AOIs are also highlighted, for instance, also highlighting the declaration of a variable when fixating on one of its uses. As the AOI connected to a variable accumulates more fixations, all the rest of the uses are also highlighted. Each group of connected AOIs is assigned one of 6 colors at load time to make them easier to distinguish. When the user switches focus, the visual feedback of the previously activated AOI decreases until it disappears completely. This strategy is reminiscent of that of Glucker et al. [Glücker et al. 2014], where an actions is triggered after a certain minimum looking duration. This strategy gives the user an immediate map over how the variable or method is accessed throughout the file, and since it relies on semantic data from a compiler it is not fooled by variables having the same name in different scopes.

Gaze data based interaction with applications needs to account for the *"Midas touch"* problem (e.g., [Jacob and Stellmach 2016; Jacob 1990]), that is, the difficulty of distinguishing intentional gaze on an item meant to trigger an action with gaze for mere visual exploration. There are a couple of ways to tackle this issue [Glücker et al. 2014; Mohan et al. 2018; Shakil et al. 2019; Velichkovsky et al. 1997], e.g., Glucker et al. [Glücker et al. 2014] required a minimum fixation duration before an action would be triggered in their EyeDE system, and in the CodeGazer system [Shakil et al. 2019] an action needs confirmation, by dwelling on a button in the periphery, before it triggers. We try to avoid spurious activations of the highlighting by gradually introducing the highlighting when the user has looked for a sufficient number of fixations.

To add this gaze-assistant on top of the GANDER platform (Section 2) we had to modify the platform in the following ways; we added analysis of fixation points (provided by the fixation service) in relation to names and relations (provided by the compiler service via the GANDER service), we added a data structure to keep track of names as AOIs and their relationships, and we added support to display the highlighting.

4 USER STUDY

To test the platform and to get user feedback on the gaze-assisted highlighting feature described in Section 3, we conducted a user study focused on the research question 'what is the user experience of doing code review with gaze-assisted highlighting of name relationships?'.

4.1 Method

4.1.1 GANDER platform setup. For this study, the GANDER platform was configured to highlight when an AOI (corresponding to a variable or method name) receives two or more fixations, including highlighting of links to the connected uses or declarations (see Fig. 3). Furthermore, if an AOI receives 30 or more fixations, the other connected AOIs are also highlighted in addition to the linking lines. If an AOI is not looked at for 3 seconds its highlighting begins

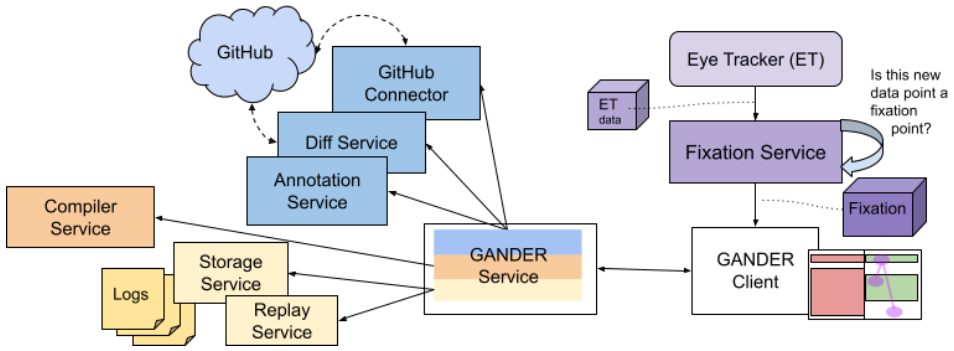

Figure 2: Architecture sketch of the platform.

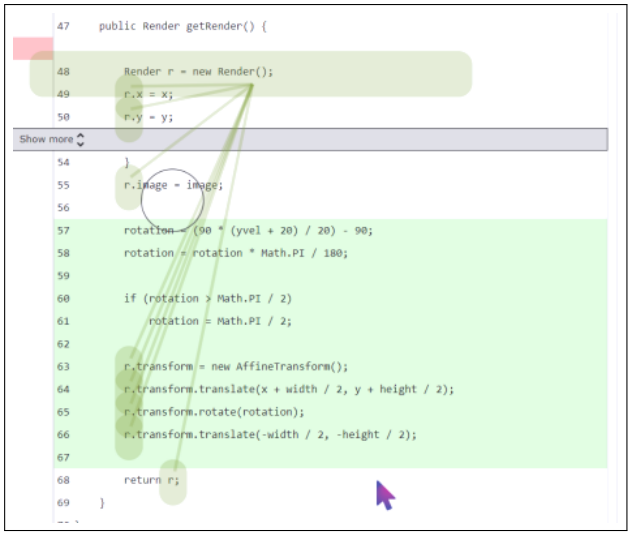

Figure 3: A screenshot of participant 6 on task 4 in replay. The AOIs are highlighted according to the rain cloud strategy.

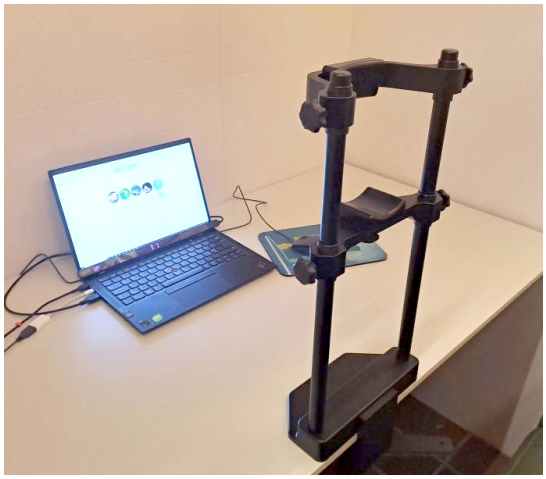

Figure 4: Picture of the experiment setup.

to fade out. As stimuli, we used pull requests from the FlappyBird repository[6]. This repository contains a game written in Java where the user controls a small bird and navigates it through a course. We chose this repository because 1) it is written in Java; 2) it is easy to explain to the participants what the app is doing and 3) it has a variety of pull requests with varying degrees of size and complexity.

4.1.2 Physical setup. To be able to recruit participants easily, we set up the study in a public area of the university trafficked by students in Computer Science and Electrical Engineering. To get closer to the setup suggested in eye-tracking experiment guidelines [Sharafi et al. 2020], we used cardboard screens (see Fig. 4) to reduce the distractions from people passing by and to give some privacy for the closing interviews. Participants' head movements were restricted with a chin and forehead rest and they viewed the stimuli on a laptop screen (32.4 cm x 21.7 cm) at a distance of 60 cm while a Tobii 4C eye tracker recorded their gaze at 90 Hz.

4.1.3 Study protocol. First, participants were provided with information about the study[7]. After providing informed consent, they were introduced to the GANDER frontend and an introduction to the FlappyBird project and its code. They were then given an initial questionnaire about previous experience with code reviews, software development and Java programming. After calibration of the eye tracker, participants were asked to complete six code review-related tasks. During the first task, they got coaching, explanations and a chance to ask questions, to make them more familiar with the user interface, the code and the structure of the tasks. For the rest of the tasks, the participant first performed the task alone and in silence and between the tasks was given the option to verbally provide a short comment about how they experienced the task and the tool. After finishing all tasks, the participants were given a closing interview to explore their experience with the GANDER platform and gaze-assisted code highlighting. The closing interview was conducted in Swedish and recorded.

4.1.4 Participants. We recruited 8 participants for the study. The first participant was used for piloting the setup and tuning the

[6]https://github.com/granttitus/FlappyBird/
[7]Study protocol and other supplementary material is available at https://portal.research.lu.se/en/publications/gander-a-platform-for-exploration-of-gaze-driven-assistance-in-co

configuration. The remaining 7 participants and their experience is listed at the top of Table 1.

4.1.5 Data collection. The data from the study was collected in a few different formats. The initial questionnaire was collected in writing via an electronic form. During the tasks, we collected gaze data and application logs in the background, as well as written notes capturing comments and reactions by the participants after completing the tasks. Finally, the closing interview was recorded as audio. We discarded the data gathered during the first training task. The first and second authors acted as study leaders during the user study. The total effective time a participant spent on the tasks varied with a mean value of 20.2 min and a median of 21.2 min.

4.1.6 Data analysis. We translated the questionnaire responses into English and entered them into tables for comparisons between participants. Then we transcribed and translated the audio recordings from the interviews, and carried out a thematic analysis. The third author created the initial themes which were discussed with the fourth and eighth authors until a consensus was reached.

4.1.7 Threats to validity. The validity of this study is limited by the small number of participants and their limited experience with code reviews, see Table 1. A more experienced code reviewer could react differently to the highlighting of the variables and find it more or less useful and distracting than the study participants. The accuracy of the eye tracker is always a limiting factor for studies with relatively small and close elements on the screen [Sharafi et al. 2020]. We optimized accuracy by stabilizing participants on a chin and forehead rest, but participants still reported that the interface did not highlight all the variables they were looking at.

4.2 Results

The results of the initial questionnaire and the thematic analysis of the closing interview are presented in Table 1. The themes constructed from the interview transcripts highlight different aspects of the experience using the platform:

- **Lack of explicit action**: All but one of the participants experienced that the interaction lacked explicit actions either for understanding how to interact with the tool (*"Difficult to understand how you were supposed to use it."* - P1, *"More confusion than help really"* - P6) or that the rain cloud highlighting disappeared unexpectedly when trying to follow the links (*"Then I tried to kind of follow where the line was going and I never had time to do that. It kind of vanished and then I had to go back"* - P2) or by wanting to use the mouse and keyboard instead (*"But what I would like personally would be something similar but where I hover with my mouse"* - P5, *"It would have been easier to just Control-F, search for it, look for it"* - P6).
- **Overwhelming**: Participants also described the experience as being overwhelming when the user interface started changing and reacting to where they were looking (*"There was a lot of things happening on the screen so it was hard to actually focus on what you wanted to find"* - P3, *"It was...a whole lot of text at the same time"* - P4).
- **Insufficient accuracy**: Problems with accuracy (*"So there were many times when I tried to look at one row and it kind of thought I was looking at the row above"* - P2, *"Look at the same spot, but different AOI lights up"* - P6) were reported by two of the participants.
- **Helpful**: Two participants found the tool helpful for navigating the code (*"I mean it was pretty nice sometimes for quickly finding where all other places where I kind of use this variable or function"* - P5, *"It was helpful"* - P1).
- **Exciting**: Being able to interact with a system using gaze and focus was described as exciting (*"It was an exciting experience!"* - P7) and novel (*"A cool thing right. It is like a bit futuristic."* - P6).

5 DISCUSSION

From the results, we find that participants experienced a lack of explicit action where it was not apparent how to start, stop or continue the highlighting. They were overwhelmed by things changing on-screen and experienced insufficient accuracy when the tool highlighted a different location than where they were looking. These are all common and expected issues in early implementations of gaze interaction systems related both to the *"Midas touch"* problem [Glücker et al. 2014; Jacob and Stellmach 2016; Jacob 1990] and to animations, especially outside your current focus, being distracting [Alzahrani et al. 2022].

That it was possible to extend the GANDER platform with eye tracking interaction and a custom highlighting pattern, and then replicate issues found in earlier studies, e.g, [Jacob and Stellmach 2016; Jacob 1990], shows that the platform demonstrates an end-to-end capability of real-time eye-tracking performance, modularity and extensibility. We think this makes GANDER an interesting test bed for further design iterations and hope it can be useful to other research and help to accelerate research into gaze-assisted code review.[8]

It is encouraging that some participants found gaze assisted code review, even in this simple format, exciting and helpful. In future iterations of the tool we would like to address the reported issues by exploring, for instance, the approaches taken in CodeGazer [Santos 2021] or DualGaze [Mohan et al. 2018], where gaze control is done in two steps. A multi-modal approach, where gaze is combined with a click on the mouse or keyboard [Glücker et al. 2014; Stellmach and Dachselt 2012], also aligns with the feedback that many participants would have liked to combine gaze with mouse and keyboard interaction. An additional direction to consider, in response to the feedback of not having enough time to follow relations, would be to detect the user becoming distracted or looking at the highlighted connections and make the interaction more adaptive by slowing down or freezing the animations for a while.

Taking a step back and considering the goals of the platform and the user study, we did not fully test all the aims of the platform. Although we tested the mobility of the setup, we did not go to an industrial context with practitioners. That considered, we did put what we estimate to be a realistic code review setup (similar design to e.g., GitHub) with good stimuli (selected from a real project on GitHub including comments by others on the selected pull requests) in front of participants and everything worked in terms of robustness, performance, and quality (with regard to the platform

[8] Open source at https://gitlab.com/lund-university/gander.

Table 1: Participants' responses to questionnaire and thematic analysis of their responses in the closing interview

		P1	P2	P3	P4	P5	P6	P7	Total
Previous experience	Java programming		✓	✓	✓	✓	✓	✓	6
	Software development		✓	✓	✓		✓	✓	5
	Code review		✓		✓		✓	✓	4
	Industry		✓				✓		2
Interview themes	Lack of explicit action	✓	✓	✓	✓	✓	✓		6
	Overwhelming			✓	✓			✓	3
	Lack of precision		✓					✓	2
	Exciting						✓	✓	2
	Helpful	✓				✓			2

behaving as intended). We were further able to use the replay mode to get a good estimate of how the participants interacted with the pull requests.

When considering the effort of adding the gaze-assisted highlighting feature, it was reasonable but could be improved. If we wanted other kinds of AOIs from the compiler service or other kinds of gaze states from the fixation service, it would have required more effort, but by using name relationships that were already available from the compiler service and limiting the gaze detection to fixations we could focus on the interaction and display. A direction to explore for the design could be to reduce the effort of adding different gaze-assistants and exposing more gaze states. In addition, it could be useful to broaden the use of data from the compiler and support languages other than Java by, for example, adding support for the Language Service Protocol [Gunasinghe and Marcus 2022]. It may further be interesting to systematically explore to what extent the different gaze metrics listed by Sharafi et al. [Sharafi et al. 2020] could be incorporated into the platform design.

6 CONCLUSIONS

We set out to design and implement a platform for user studies in gaze-assisted code review incorporating near real-time analysis of gaze data, enabling implementation of a gaze-assistant that highlights name relationships in the code based on gaze behavior. We then evaluated the gaze-assistant in a user study with 7 participants. Our results indicate that the participants experienced, e.g., lack of explicit action and being overwhelmed, to a large degree, but also excitement. While these results align with results in earlier work, we see the completed user study as a testament to the platform's capabilities with regard to mobility, real-time gaze interaction, data logging, replay and analysis. We see several possible directions for future work, for instance, continued refinement of the platform to enable assistance for other languages based on more kinds of AOIs and gaze states, as well as, further refinements of gaze-driven and user intention-aware provision of contextual information. Another interesting direction is adding communication between distributed GANDER clients for research into collaborative eye tracking [Cheng et al. 2021].

ACKNOWLEDGMENTS

The authors would like to thank Christofer Rydenfält, Lund University, for feedback during the preparation of the user study. The authors would further like to thanks the following funders who partly funded this work: the Swedish strategic research environment ELLIIT, the Swedish Foundation for Strategic Research (grant nbr. FFL18-0231), the Swedish Research Council (grant nbr. 2019-05658), and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

REFERENCES

Mona Alzahrani, Alexandra L. Uitdenbogerd, and Maria Spichkova. 2022. Impact of animated objects on autistic and non-autistic users. In *The 44th International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE/ACM, 102–112. https://doi.org/10.1145/3510458.3513007

Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

Andrew Begel and Hana Vrzakova. 2018. Eye Movements in Code Review. In *Workshop on Eye Movements in Programming (EMIP)*. Article 5, 5 pages.

Shiwei Cheng, Jialing Wang, Xiaoquan Shen, Yijian Chen, and Anind Dey. 2021. Collaborative eye tracking based code review through real-time shared gaze visualization. 16, 3 (2021), 163704. https://doi.org/10.1007/s11704-020-0422-1

Antonio Diaz-Tula and Carlos H Morimoto. 2017. Robust, real-time eye movement classification for gaze interaction using finite state machines. In *2017 COGAIN symposium*.

Torbjörn Ekman and Görel Hedin. 2007. The jastadd extensible java compiler. In *Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems, languages and applications*. 1–18.

Hartmut Glücker, Felix Raab, Florian Echtler, and Christian Wolff. 2014. EyeDE: gaze-enhanced software development environments. In *CHI'14 Extended Abstracts on Human Factors in Computing Systems*. 1555–1560.

Drew T Guarnera, Corey A Bryant, Ashwin Mishra, Jonathan I Maletic, and Bonita Sharif. 2018. itrace: Eye tracking infrastructure for development environments. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*. 1–3.

Nadeeshaan Gunasinghe and Nipuna Marcus. 2022. *Language Server Protocol and Implementation: Supporting Language-Smart Editing and Programming Tools*. Apress. https://doi.org/10.1007/978-1-4842-7792-8

Rob Jacob and Sophie Stellmach. 2016. What You Look at is What You Get: Gaze-Based User Interfaces. *Interactions* 23, 5 (aug 2016), 62–65. https://doi.org/10.1145/2978577

Robert J. K. Jacob. 1990. What You Look at is What You Get: Eye Movement-Based Interaction Techniques. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Seattle, Washington, USA) *(CHI '90)*. Association for Computing Machinery, New York, NY, USA, 11–18. https://doi.org/10.1145/97243.97246

Peng Kuang, Emma Söderberg, Diederick C. Niehorster, and Martin Höst. 2023. Towards Gaze-Assisted Developer Tools. In *International Conference on Software Engineering: New Ideas and Emerging Results (ICSE-NIER)*. IEEE.

Pallavi Mohan, Wooi Boon Goh, Chi-Wing Fu, and Sai-Kit Yeung. 2018. DualGaze: Addressing the Midas Touch Problem in Gaze Mediated VR Interaction. In *2018 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*. 79–84. https://doi.org/10.1109/ISMAR-Adjunct.2018.00039

Diederick C. Niehorster, Richard Andersson, and Marcus Nyström. 2020. Titta: A toolbox for creating PsychToolbox and Psychopy experiments with Tobii eye trackers. *Behavior Research Methods* 52, 2 (2020), 1970–1979. https://doi.org/10.3758/s13428-020-01358-8

Unaizah Obaidellah, Mohammed Al Haek, and Peter C.-H. Cheng. 2018. A Survey on the Usage of Eye-Tracking in Computer Programming. *ACM Comput. Surv.* 51, 1 (2018).

Stevche Radevski, Hideaki Hata, and Kenichi Matsumoto. 2016. EyeNav: gaze-based code navigation. In *Proceedings of the 9th Nordic Conference on Human-Computer Interaction*. 1–4.

Caitlin Sadowski, Emma Söderberg, Luke Church, Michal Sipko, and Alberto Bacchelli. 2018. Modern code review: a case study at google. In *Proceedings of the 40th international conference on software engineering: Software engineering in practice*. 181–190.

André L Santos. 2021. Javardeye: Gaze Input for Cursor Control in a Structured Editor. In *Companion Proceedings of the 5th International Conference on the Art, Science, and Engineering of Programming*. 31–35.

Asma Shakil, Christof Lutteroth, and Gerald Weber. 2019. Codegazer: Making code navigation easy and natural with gaze input. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–12.

Zohreh Sharafi, Bonita Sharif, Yann-Gaël Guéhéneuc, Andrew Begel, Roman Bednarik, and Martha Crosby. 2020. A practical guide on conducting eye tracking studies in software engineering. *Empirical Software Engineering* 25 (2020), 3128–3174.

Zohreh Sharafi, Zéphyrin Soh, and Yann-Gaël Guéhéneuc. 2015. A systematic literature review on the usage of eye-tracking in software engineering. *Information and Software Technology* 67 (2015), 79–107.

Emma Söderberg, Luke Church, Jürgen Börstler, Diederick Niehorster, and Christofer Rydenfält. 2022. Understanding the Experience of Code Review: Misalignments, Attention, and Units of Analysis *(EASE '22)*. 170–179.

Sophie Stellmach and Raimund Dachselt. 2012. Look & touch: gaze-supported target acquisition. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 2981–2990.

Tamara Van Gog, G.W.C. Paas, J.J.G. van Merrienboer, and P. Witte. 2005. Uncovering the Problem-Solving Process: Cued Retrospective Reporting Versus Concurrent and Retrospective Reporting. *Journal of Experimental Psychology: Applied* 11, 4 (Dec. 2005), 237–244. https://doi.org/10.1037/1076-898X.11.4.237

Boris Velichkovsky, Andreas Sprenger, and Pieter Unema. 1997. Towards gaze-mediated interaction: Collecting solutions of the "Midas touch problem". In *Human-Computer Interaction INTERACT '97: IFIP TC13 International Conference on Human-Computer Interaction, 14th–18th July 1997, Sydney, Australia*, Steve Howard, Judy Hammond, and Gitte Lindgaard (Eds.). Springer US, Boston, MA, 509–516. https://doi.org/10.1007/978-0-387-35175-9_77

Hana Vrzakova, Andrew Begel, Lauri Mehtätalo, and Roman Bednarik. 2020. Affect Recognition in Code Review: An In-situ Biometric Study of Reviewer's Affect. *Journal of Systems and Software* 159 (2020), 110434.