Humans in Empirical Software Engineering Studies: An Experience Report

Bonita Sharif and Niloofar Mansoor
School of Computing
University of Nebraska - Lincoln
Lincoln, Nebraska USA
bsharif@unl.edu, niloofar@huskers.unl.edu

***Abstract*—The use of human validation in software engineering methods, tools, and processes is crucial to understanding how these artifacts actually impact the people using them. In this paper, we report our experiences on two methods of data collection we have used in software engineering empirical studies, namely online questionnaire-based data collection and in-person eye tracking data collection using eye tracking equipment. The design and instrumentation challenges we faced are discussed with possible ways to mitigate them. We conclude with some guidelines and our vision for the future in human-centric studies in software engineering.**

***Index Terms*—empirical studies, eye tracking, software engineering, surveys, experiences, guidelines**

I. Introduction

Building software is a complex phenomenon. The developers that produce software artifacts are also their consumers. As developers, we work not only with the artifacts (code, design, test cases, requirements, etc...) we create, but also with artifacts created by other developers across teams. In order to increase comprehension [1], happiness [2] and productivity [3] in dealing with such artifacts, it is important that they are written in a way that is intuitive to produce and consume. In order to better serve developers and related stakeholders, it is crucial to conduct empirical studies in specific contexts, to better understand cause and effect in certain situations and tasks when developers work with artifacts.

The results of such experiments should feed back into the production of software artifacts. However, this feedback loop does not happen with just one experiment. It takes a set of experiments over many years to determine the factors influencing what we seek to change. This is why replication is crucial. Replications [4] (exact or inexact) help in understanding if the results still hold for other groups and demographics of people, and if not, what individual differences exist. The emergence of the replication and negative (RENE) results tracks at major software engineering conferences is evidence that we are moving in the right direction to build more meaningful theories [5]–[8] in comprehension of software engineering research and practice. A lot of the theories in program comprehension stem from the social sciences and while some of them might hold, others might not. This is due to the fact that building software and the interactions that come from it might not always adhere to other social norms that exist. Empirical studies are one evidence-based way to move the relatively young field of software engineering and its theories forward. A great example of how to improve our use of theory for computing education research was given by Nelson and Ko [9]. The program comprehension sub-field of software engineering can also benefit from their line of thought while designing experiments.

Our research team has conducted several eye tracking [10] empirical studies [11]–[16] (exploratory and controlled experiments, in industry and academia) since 2005. Because eye tracking was a relatively new application in software engineering at the time, we received many early rejections because the technology and framework, now referred to as iTrace [17] (see http://www.i-trace.org/), was not believable. This is somewhat understandable since the peer review process depends on experts in the field. If your work is a relatively new interdisciplinary field, the typical expert in the field would not be fully aware of what the technology can or cannot do and hence are bound to be skeptical (or overly accepting - which is also detrimental) about it. The burden of presenting convincing arguments is always on the author. The initial feedback we received helped to reshape our motivation on the use of eye tracking in software engineering. The peer review system is not perfect but certainly helped refine our initial ideas when only a handful of people in our field were doing eye tracking studies. One of the very first studies that showed the value of eye tracking on source code in contrast to keyboard and mouse interaction was our 2015 paper with Fritz and Shepherd [14], [18], where we collected eye tracking data using iTrace and interaction data using Mylyn simultaneously. This study paved the way to make eye tracking believable, feasible, and realistic as it was a working prototype showing value in the rich finer-grained data eye tracking produced compared to the keyboard and mouse interaction logs. Since 2006, many more software engineering researchers have entered the field of eye tracking and program comprehension. This has resulted in more people in the field able to contribute to the review process of interdisciplinary papers.

In the rest of this paper, we discuss the need for multiple modes of data collection and focus on two main ones our team has been involved with: eye tracking studies and online questionnaire-based studies. We then present some community guidelines by our team and the community at large and conclude with our vision for human-centric studies in software engineering.

II. The Need for Multiple Data Collection Modes

There are many ways in which a researcher can go about collecting data for experiments [19]. There is no one method of collection that is better than another. There might be one that is more suitable (for the specific research question) than another. Each method can give insights about a different aspect of what is being studied. There are advantages and disadvantages to every method and this should be considered during the study design phase. For example, just because you have an eye tracker does not mean you should conduct an eye tracking study. In fact, not all studies we conduct in our lab are eye tracking studies. Sometimes, they make sense depending on what you seek to measure. For example, if you are looking to determine what particular lines or words developers actually looked at while doing a task, it would make sense to use an eye tracker as a method of data collection. However, just doing this is not enough, the researcher also needs to use the right metrics [20] to keep internal validity high.

One strategy used in our team is to conduct the same study online and using an eye tracker. The same study is replicated with different participants but same tasks only using a different mode of data collection. It is extremely time and labor intensive to conduct an eye tracking study. It has to be done one participant at a time with a moderator present (to check for calibration, positioning and starting/stopping the tracking) and they usually last about an hour or two hours (for a realistic set of tasks) from start to finish. They are also typically done over the course of many more months compared to online studies. Since the online questionnaire based study can be deployed once to many people, many more participants can be recruited. We report on our experiences in using Mechanical Turk later in the paper. One advantage of doing the same study online and in person using an eye tracker is that you can gain further insight into some results generated from the online study. For example, you would be able to further tell the thought processes of participants with similar answers (from the online study).

In summary, instead of thinking of just one mode of data collection, consider multiple sources if possible. This does not have to be an eye tracking study. It could be a retrospective interview, full interview-based study, verbal summaries [21], or a video-based study where the screen is being recorded while developers perform tasks.

In the next two sections, we present some of our experiences in conducting eye tracking studies and online questionnaire-based studies in software engineering.

III. Eye Tracking Studies

There has been a surge in eye tracking studies [22] in software engineering since 2006. Even though eye tracking has been around since the 1800's, it is only recently being used by more software engineering researchers. For designing and conducting eye tracking studies in software engineering, we direct the reader to a one-stop practical guide [23]. One of the first studies was done by Crosby and Stelovsky in 1990 [24]. They looked at how participants read the binary search algorithm in Pascal code and analyzed the scan patterns at two experience levels. They found that all nineteen participants viewed most areas of the algorithm compared to natural text (a finding we also were able to reproduce in 2015 [25]). They also found that the low experience group spent more time reading comments over code compared to the high experience group. Overall, they found experienced programmers recognize more meaningful areas of the code and spend more time there. They call for more studies on such expertise classification strategies.

The Crosby study and most eye tracking studies in software engineering are done on short code snippets. The reason is that there was no infrastructure to support viewing longer code snippets. In order to understand why this was a problem, we need to first understand how an eye tracker works. Research-grade eye trackers work by presenting an image or text (i.e., stimuli) on a computer screen, and then use the data from cameras to determine the location (x,y coordinate) the person is looking at. One limitation of current state-of-the-art eye trackers and vendor software is that they only work on fixed stimuli (i.e., an image or text) that fits on the computer screen. Changes to the stimuli (screen), such as scrolling, present a very complex problem. There is no existing support (in any commercial eye trackers) for a subject to interactively use an editor or switch between different files. Basically, existing systems do not keep track of what line in which file is present on the screen (i.e., currently being viewed). In order to circumvent this problem, we introduced iTrace [17], community eye tracking infrastructure to support conducting large-scale realistic studies. A detailed explanation of why current state of the art methods do not work for realistic software tasks is given in Shaffer et al. [26]. A detailed technical briefing [27] for iTrace was held by the iTrace team in 2016 at the ICSME conference in Raleigh, NC to introduce eye tracking limitations to the broader software engineering community. Next, we briefly explain iTrace and then present some challenges researchers face when conducting eye tracking studies.

A. *iTrace - Eye Tracking Community Infrastructure*

iTrace is community eye tracking infrastructure that makes it possible for software engineering researchers to conduct eye tracking studies on large software systems right within the IDE. Using iTrace eliminates the limitation of using short code snippets as images/short text to view on the screen. iTrace is designed as a plugin-based client/server architecture. It runs as a plugin inside the Integrated Development environment. The first prototype of iTrace was made available in 2015 and only supported Eclipse. Since then, we completely redesigned the infrastructure and now there is support for Eclipse, MS Visual Studio, Atom, and the Chrome browser. In 2020, we started supporting high speed eye trackers in iTrace with the release of Déjà Vu [28]. This was essential if we wanted to use eye tracking over 60Hz in the IDE because as the number of eye samples per second increased beyond 60 samples per second, the system was not able to map the real time gaze to the correct element being looked at due to the response time limitations of Windows function calls. We worked around this by doing all of this mapping in a post-processing step by recording every event and replaying it at a slower speed using various delay strategies [28]. Please refer to http://www.i-trace.org/ for future downloads of iTrace and its plugins.

B. Challenges

There are many challenges in conducting eye tracking studies, some of which are beyond our control. For example, there is a small subset of the population that can never be tracked due to the way their eye anatomy is structured. Fortunately, eye tracking equipment has come a long way. Today, one can purchase a state of the art research grade tracker that looks like a bar that clips under the monitor. This was not the case in the 1970's. We only expect this technology to get better in the future. In fact, we envision every laptop to come embedded with an eye tracker in the foreseeable future.

1) Design and Metrics: One of the main difficulties in eye tracking studies come with calibration. It is extremely important to calibrate each person before each task. This process has dramatically improved as eye tracking equipment has gotten better. The lighting and positioning all need to be carefully set before any real data is collected. It is important to do pilots for this purpose. In our lab, we take at least 6 months to design and pilot studies before actual data collection begins. As mentioned before, the time and effort it takes to bring in participants is quite high and we need to make sure that our system is working as intended before starting the actual study. By piloting, we refer to not only testing if the data collection works as expected but also putting the data through the entire process of data analysis as well. This means, all the threats to validity are identified, dependent variables, and eye metrics [20] one plans to use are determined before any data collection begins. For example, if using pupil dilation as a metric, it is crucial that you use a chin rest with your study data collection as this measure is affected both by movement and by lighting changes on the monitor and the room. It is not wise to decide to use pupil dilation (this metric is still output by most trackers in the raw data generated) after the fact when the data collection did not use a chin rest nor was the lighting of room and monitor factored into the design setup. Another important challenge for any study is choosing the correct tasks. Task selection can make or break your study. Again, this is why we do pilots.

2) Webcams as eye trackers: Recently, due to the COVID-19 pandemic, many labs were forced to shut down (including our own). The push for using webcams as eye trackers so eye-tracking studies can be done remotely was promoted by companies offering these services. As more companies, such as RealEye [29], offer webcam eye tracking as a solution to the high-cost of screen-based eye trackers, it is important to compare them to determine how viable this method of collecting gaze data is. As a test in our lab, we ran 5 participants, who analyzed 3 code snippets and watched 3 short movie clips, with a webcam tracker (WebGazer [30]) and research grade eye tracker (Tobii TX-300) tracking participants at the same time. The entire study was done in iTrace-Chrome. Initial results show that even with continuous calibration, the webcam used as a tracker doesn't present results suitable for research analysis and we would not recommend it currently as a suitable substitute. Previous studies have compared low-cost eye trackers [31], [32], however these studies have focused only on comparing purpose-built eye trackers with each other and have failed to compare webcam trackers. While the accuracy of webcam trackers have been evaluated alongside their introduction into the literature [30], the effectiveness and actual usability of these devices is still largely untested. Pinpointing the level of accuracy they provide can help future researchers determine the level of tracker best suited for their project.

3) Editing in Eye tracking studies: Editing in eye tracking studies presents an issue during tracking as the words are constantly changing as an edit is made. Fakhoury et al. introduced a solution for this in iTrace-Atom [33]. However, a more comprehensive solution is needed to support all other plugins. This is not as trivial as it seems as all possible scenarios for editing need to be considered including copy and pasting large code. The iTrace team along with some community collaborators are working towards a usable solution in the near future. One step towards this solution is the Déjà Vu [28] approach which takes all processing of gaze mapping offline. During gaze replay and mapping, the editing issue will be addressed in future infrastructure releases. However, at this time, iTrace does not support editing across its plugins.

IV. Online Studies

This section presents some discussion on designing surveys using Qualtrics [34], using Mechanical Turk and challenges with hosting and crowdsourcing online studies.

A. Designing Surveys Using Qualtrics

Qualtrics [34] is a powerful software solution for hosting and distributing surveys and collecting data for research. It provides an online environment in which even novice users can build useful surveys with simple or complex logic. Qualtrics features many different question types, question randomization options, complex display logic, bot detection, and customizable looks and designs for surveys. It also provides custom coding features so that survey designers can use JavaScript to add advanced functionality to their surveys.

We have used Qualtrics to host and design some of our online software engineering studies, in which we have shown code snippets to participants and asked different types of questions about the code. We found that Qualtrics comes with a lot of useful standard features, but we needed additional features in order to publish a survey that was mostly coding questions. The platform's basic features make it easy to provide an informed consent form to participants, show textual and graphical information, assign IDs to participants, and determine the display logic for the survey. In addition to storing the answers that the users give throughout the survey, Qualtrics can also save timing data and some interaction data (e.g. number of clicks on a question page). Users can also pause working on the survey and get back to it later on. It is also simple to limit answers to specific types (e.g. only allowing an email address to be entered in a text field that asks for an email address), as well as enforcing answers on questions that are mandatory. In our experience, modifying the survey flow and display logic was very straightforward, and we could easily implement our intended logic and randomly show different participants different treatments in code snippets.

Since we were trying to show the needed code in questions as realistically and as close to how developers see code everyday, we faced some challenges in working with Qualtrics. In one of our surveys, we had questions in which the participants needed to look at multiple files to answer a question. Developers usually use IDEs when working on coding projects, and usually each file is opened as a tab and can be used by the developer. We decided to include a tabbed view inside the questions that required traversal of multiple files, and we soon realized that due to the limitations of using JavaScript in Qualtrics, the tabbed views were not working as smoothly as expected. We ended up creating pages that included the tabbed views on a hosting platform (in our case we used AWS S3) and then embedded the pages into our questions. We also felt it was necessary to add syntax highlighting to the code in our questions, and we found that adding our CSS for syntax highlighting to Qualtrics was very straightforward as well. Additionally, there were some tasks that did not fit with the classic survey format. We had to design separate web applications for them, and asked our participants to click on external links and enter their Qualtrics ID in the app so that their data can be connected. Asking the participants to manually enter their ID in another external webpage brought some issues, and we believe that Qualtrics can benefit from a functionality that makes for easier communication with other web applications. And finally, while the fraud detection capabilities works for well to catch people who are trying to participate in a survey using a single IP address, or bots who are spamming the survey, it cannot stop participants who use VPNs or other tools to mask their IPs and participate again. Sifting through spam submissions is a major challenge, and it takes a considerable amount of time and energy from the researcher. However, it is important to do because the integrity of the data is threatened if the spam submissions are not deleted.

B. Crowdsourcing Using Mechanical Turk

Amazon Mechanical Turk (MTurk) [35] is a crowdsourcing platform with which "Requesters" (employers) can hire remotely located "Workers" (contractors) to do on-demand tasks. A Human Intelligence Task, or HIT, is a self-contained task that a requester can create and a worker can work on, submit, and collect a reward for completing [36]. The MTurk website gets a fee for each HIT that is completed by the worker. The requester can choose to accept or reject a submitted HIT, and has a chance to explain to the worker why they chose to reject their submission. Since we hosted our survey on Qualtrics, we needed to be able to identify which workers completed the Qualtrics survey. So we generated a random ID for each participant that completed the Qualtrics survey, and asked for the ID in the MTurk task page. For additional reliability, we also asked for the MTurk worker ID in our Qualtrics survey. We posted our HIT as a webpage that included the instructions for the survey, the Qualtrics survey link, and a text box for the workers to provide their Qualtrics ID. One thing we noticed while using MTurk was that despite the fact that a lot of surveys that include programming questions in specific languages are posted on the website, the website only has broad categories for choosing target workers (e.g. we can't choose Java developers as our target population, only workers whose employment industry is "Software and IT Services"). This can have an effect on the quality of the data we collect, as some programmers are not experienced with the specific language or tool the survey is asking questions about. We got around this by building a proficiency test into our Qualtrics questionnaire to weed out people without the skill set we wanted. We believe that creating tags for skills would be a nice addition to the MTurk system and can improve the quality of data for the posted surveys. Another issue we faced was the number of spam submissions. Even though the MTurk worker IDs were different we noticed that some answers in our survey were copy and pasted, and their IPs on the Qualtrics result were similar. It would be beneficial for researchers and employers if MTurk can devise a way to minimize spamming and fraud, as it can affect data integrity and complicated data cleaning and analysis.

C. Challenges

Looking back at our experience with hosting surveys on Qualtrics and crowdsourcing using MTurk, the most important challenges we faced were working around Qualtrics' customizability issues for building webpages that included code, and spam and fraud detection. Qualtrics does not allow to add scripts that are placed outside of its own specific JavaScript functions that can be modified, and it constantly removed our added scripts. We worked around it by hosting our tabbed pages that included code with syntax highlighting on another website, and embedded them in our Qualtrics questions. We also had to work around some limitations for creating specific tasks by building apps and putting a link for them on our Qualtrics survey for our participants to complete those tasks. We also had issues with spam and fraud detection on both MTurk and Qualtrics, and finding solutions to disregard spam data proved to be very challenging for us. We eventually had to examine the timing data and determine a duration range in which an individual would complete a task based on the difficulty of that task. We realize that this is not a guaranteed method and we might have ended up losing some viable data, but there are no hard and fast rules for determining spam and it needs to be done by creating specific criteria related to the research study at hand.

V. Community and Guidelines

The ACM SigPlan Checklist [37] for empirical studies is a good one stop checklist to determine if your study holds up to important criteria. It also provides a good reviewing checklist for reviewers. If conducting eye tracking studies in software engineering, we recommend using iTrace [17] (see http://www.i-trace.org/) and checking out the practical guide [23] on how to design and conduct such studies. Some journals require you to follow certain templates when reporting results in studies. We have used one such model, namely CONSORT in one of our recent studies [38]. Using such templates takes out the guesswork on what to include in your writeup and also helps reviewers during the review cycle. We hope more journals require studies to have at least specific sections if they are of a certain empirical nature. We realize that not all empirical studies can adhere to one template.

Finally, if possible we suggest that authors provide an anonymised replication package for the study (tasks, scripts, code, instructions, data) so others may replicate it. This package should be created during the process of conducting the study not after the fact. Often times, if the student responsible for collecing the data graduates, it becomes hard for the PI responsible for the project to find all the pieces of data to package together. This is why it is crucial to create the data sets and replication packages while the study is active and not after the fact. We have had several experiences where the student after graduating never completed the data set curation to serve as a replication package. Due to this past experience, we now provide a replication package during the first review of the publication which makes it easier on everyone once the project is complete. Frameworks like osf.io [39], figshare [40], and zenodo.org [41] are great options for this. As an example, refer to the replication package [42] for our eye tracking study on code summarization [43].

VI. Our Vision for Human-Centric Studies

In the past 10 years, there has been a lot of emphasis on conducting studies with humans. Besides, eye tracking, some researchers like Fakhoury et al. [44] used eye tracking and brain imaging to study the effects of poor lexicon in source code on program comprehension and readability. Their results showed that poor naming and documentation practices increase the cognitive load in developers while solving tasks. Such interdisciplinary research should be welcomed in the software engineering community. But this also comes with some inherent difficulties in reviewing. The need for interdisciplinary reviewers becomes important in this case. A short synopsis of our vision of eye tracking in software engineering research and practice was presented at the ESEC/FSE Visions track [45].

We envision future studies to branch out into different tasks that software developers perform (not just summarization - as most studies in eye tracking are on summarization tasks). Our team has done studies in bug fixing (in open source systems), find the output, and summarization. However, these are obviously a very small subset of tasks developers perform. We encourage researchers to expand the types of tasks tested in empirical studies using humans. For example, How are testers actually testing the code? How are requirements and user stories interacted with? Branching out into testing other language paradigms is also important to get a full picture of how comprehension differs between expertise and paradigms.

Finally, it is important to report studies in the context they were done in and not generalize them to all situations. Each study is a small drop of evidence and does not provide complete proof of any one concept. It is one step towards learning more about the complex beings we are as developers (and humans) as we do our best to produce and consume better software.

VII. Conclusions

In this paper, we describe our experiences in conducting empirical studies in person using eye tracking equipment and online using questionnaires via Qualtrics and Mechanical Turk. Conducting studies is a time consuming and labor intensive process. There is no perfect system to conduct a study and nor is there a perfect study. We strive to achieve the best control we can in the circumstances provided, taking care to maintain the validity of the study without jeopardizing the effect. We hope that our experiences presented here, our vision and guidelines are useful to further improve the state of the art as it currently stands.

Acknowledgments

This work is supported by the US National Science Foundation under Grant Numbers CNS 18-55753 and CCF 18-55756

References

[1] M. Wyrich, A. Preikschat, D. Graziotin, and S. Wagner, "The mind is a powerful place: How showing code comprehensibility metrics influences code understanding," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 512–523. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00055

[2] D. Graziotin, F. Fagerholm, X. Wang, and P. Abrahamsson, "What happens when software developers are (un)happy," *J. Syst. Softw.*, vol. 140, pp. 32–47, 2018. [Online]. Available: https://doi.org/10.1016/j.jss.2018.02.041

[3] N. Shrestha, C. Botta, T. Barik, and C. Parnin, "Here we go again: Why is it difficult for developers to learn another programming language?" in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 691–701. [Online]. Available: https://doi.org/10.1145/3377811.3380352

[4] A. A. Neto, "A strategy to support replications of controlled experiments in software engineering," *SIGSOFT Softw. Eng. Notes*, vol. 44, no. 3, p. 23, nov 2019. [Online]. Available: https://doi.org/10.1145/3356773.3356796

[5] A. Von Mayrhauser and A. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, pp. 44–55, 1995.

[6] M.-A. Storey, "Theories, methods and tools in program comprehension: past, present and future," in *13th International Workshop on Program Comprehension (IWPC'05)*, 2005, pp. 181–191.

[7] J. Koenemann and S. P. Robertson, “Expert problem solving strategies for program comprehension,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1991, pp. 125–130.

[8] T. D. LaToza, D. Garlan, J. D. Herbsleb, and B. A. Myers, “Program comprehension as fact finding,” in *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2007, pp. 361–370.

[9] G. L. Nelson and A. J. Ko, “On use of theory in computing education research,” in *Proceedings of the 2018 ACM Conference on International Computing Education Research, ICER 2018, Espoo, Finland, August 13-15, 2018*, L. Malmi, A. Korhonen, R. McCartney, and A. Petersen, Eds. ACM, 2018, pp. 31–39. [Online]. Available: https://doi.org/10.1145/3230977.3230992

[10] B. Sharif and T. Shaffer, “The use of eye tracking in software development,” in *Foundations of Augmented Cognition - 9th International Conference, AC 2015, Held as Part of HCI International 2015, Los Angeles, CA, USA, August 2-7, 2015, Proceedings*, ser. Lecture Notes in Computer Science, D. Schmorrow and C. M. Fidopiastis, Eds., vol. 9183. Springer, 2015, pp. 807–816. [Online]. Available: https://doi.org/10.1007/978-3-319-20816-9_77

[11] N. A. Madi, C. S. Peterson, B. Sharif, and J. I. Maletic, “From novice to expert: Analysis of token level effects in a longitudinal eye tracking study,” in *29th IEEE/ACM International Conference on Program Comprehension, ICPC 2021, Madrid, Spain, May 20-21, 2021*. IEEE, 2021, pp. 172–183. [Online]. Available: https://doi.org/10.1109/ICPC52881.2021.00025

[12] J. A. Saddler, C. S. Peterson, S. Sama, S. Nagaraj, O. Baysal, L. Guerrouj, and B. Sharif, “Studying developer reading behavior on stack overflow during API summarization tasks,” in *27th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2020, London, ON, Canada, February 18-21, 2020*, K. Kontogiannis, F. Khomh, A. Chatzigeorgiou, M. Fokaefs, and M. Zhou, Eds. IEEE, 2020, pp. 195–205. [Online]. Available: https://doi.org/10.1109/SANER48275.2020.9054848

[13] S. Aljehane, B. Sharif, and J. Maletic, “Determining differences in reading behavior between experts and novices by investigating eye movement on source code constructs during a bug fixing task,” in *ACM Symposium on Eye Tracking Research and Applications*, ser. ETRA ’21 Short Papers. New York, NY, USA: Association for Computing Machinery, 2021. [Online]. Available: https://doi.org/10.1145/3448018.3457424

[14] K. Kevic, B. Walters, T. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz, “Eye gaze and interaction contexts for change tasks - observations and potential,” *J. Syst. Softw.*, vol. 128, pp. 252–266, 2017. [Online]. Available: https://doi.org/10.1016/j.jss.2016.03.030

[15] C. Peterson, J. Saddler, T. Blascheck, and B. Sharif, “Visually analyzing students’ gaze on c++ code snippets,” in *EMIP 2019-6th International Workshop on Eye Movements in Programming*, 2019.

[16] T. Blascheck and B. Sharif, “Visually analyzing eye movements on natural language texts and source code snippets,” in *ETRA 2019-ACM Symposium on Eye Tracking Research & Applications*, 2019.

[17] D. T. Guarnera, C. A. Bryant, A. Mishra, J. I. Maletic, and B. Sharif, “itrace: eye tracking infrastructure for development environments,” in *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*. ACM, 2018, p. 105.

[18] K. Kevic, B. M. Walters, T. R. Shaffer, B. Sharif, D. C. Shepherd, and T. Fritz, “Tracing software developers’ eyes and interactions for change tasks,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2015. New York, NY, USA: Association for Computing Machinery, 2015, p. 202–213. [Online]. Available: https://doi.org/10.1145/2786805.2786864

[19] C. Wohlin, M. Höst, and K. Henningsson, “Empirical research methods in software engineering,” in *Empirical Methods and Studies in Software Engineering, Experiences from ESERNET*, ser. Lecture Notes in Computer Science, R. Conradi and A. I. Wang, Eds., vol. 2765. Springer, 2003, pp. 7–23. [Online]. Available: https://doi.org/10.1007/978-3-540-45143-3_2

[20] Z. Sharafi, T. Shaffer, B. Sharif, and Y. Guéhéneuc, “Eye-tracking metrics in software engineering,” in *2015 Asia-Pacific Software Engineering Conference, APSEC 2015, New Delhi, India, December 1-4, 2015*, J. Sun, Y. R. Reddy, A. Bahulkar, and A. Pasala, Eds. IEEE Computer Society, 2015, pp. 96–103. [Online]. Available: https://doi.org/10.1109/APSEC.2015.53

[21] S. Letovsky, “Cognitive processes in program comprehension,” *Journal of Systems and software*, vol. 7, no. 4, pp. 325–339, 1987.

[22] U. Obaidellah, M. Al Haek, and P. C.-H. Cheng, “A survey on the usage of eye-tracking in computer programming,” *ACM Comput. Surv.*, vol. 51, no. 1, pp. 5:1–5:58, Jan. 2018. [Online]. Available: http://doi.acm.org/10.1145/3145904

[23] Z. Sharafi, B. Sharif, Y. Guéhéneuc, A. Begel, R. Bednarik, and M. E. Crosby, “A practical guide on conducting eye tracking studies in software engineering,” *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 3128–3174, 2020. [Online]. Available: https://doi.org/10.1007/s10664-020-09829-4

[24] M. E. Crosby and J. Stelovsky, “How do we read algorithms? A case study,” *Computer*, vol. 23, no. 1, pp. 24–35, 1990. [Online]. Available: https://doi.org/10.1109/2.48797

[25] T. Busjahn, R. Bednarik, A. Begel, M. Crosby, J. H. Paterson, C. Schulte, B. Sharif, and S. Tamm, “Eye movements in code reading: Relaxing the linear order,” in *2015 IEEE 23rd International Conference on Program Comprehension*. IEEE, 2015, pp. 255–265.

[26] T. R. Shaffer, J. L. Wise, B. M. Walters, S. C. Müller, M. Falcone, and B. Sharif, “itrace: enabling eye tracking on software artifacts within the IDE to support software engineering tasks,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Bergamo, Italy, August 30 - September 4, 2015*, E. D. Nitto, M. Harman, and P. Heymans, Eds. ACM, 2015, pp. 954–957. [Online]. Available: https://doi.org/10.1145/2786805.2803188

[27] B. Sharif and J. I. Maletic, “itrace: Overcoming the limitations of short code examples in eye tracking experiments,” in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2016, pp. 647–647.

[28] V. Zyrianov, D. T. Guarnera, C. S. Peterson, B. Sharif, and J. I. Maletic, “Automated recording and semantics-aware replaying of high-speed eye tracking and interaction data to support cognitive studies of software engineering tasks,” in *IEEE International Conference on Software Maintenance and Evolution, ICSME 2020, Adelaide, Australia, September 28 - October 2, 2020*. IEEE, 2020, pp. 464–475. [Online]. Available: https://doi.org/10.1109/ICSME46990.2020.00051

[29] “Realeye.” [Online]. Available: https://www.realeye.io/

[30] A. Papoutsaki, P. Sangkloy, J. Laskey, N. Daskalova, J. Huang, and J. Hays, “Webgazer: Scalable webcam eye tracking using user interactions,” in *Proceedings of the 25th International Joint Conference on Artificial Intelligence (IJCAI)*. AAAI, 2016, pp. 3839–3845.

[31] G. Funke, E. Greenlee, M. Carter, A. Dukes, R. Brown, and L. Menke, “Which eye tracker is right for your research? performance evaluation of several cost variant eye trackers,” in *Proceedings of the Human Factors and Ergonomics Society annual meeting*, vol. 60, no. 1. SAGE Publications Sage CA: Los Angeles, CA, 2016, pp. 1240–1244.

[32] J. Titz, A. Scholz, and P. Sedlmeier, “Comparing eye trackers by correlating their eye-metric data,” *Behavior Research Methods*, vol. 50, no. 5, pp. 1853–1863, 2018.

[33] S. Fakhoury, D. Roy, H. Pines, T. Cleveland, C. S. Peterson, V. Arnaoudova, B. Sharif, and J. Maletic, “gazel: Supporting source code edits in eye-tracking studies,” in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 2021, pp. 69–72.

[34] “Qualtrics.” [Online]. Available: https://www.qualtrics.com/

[35] “Amazon mechanical turk.” [Online]. Available: https://www.mturk.com/

[36] “Amazon mechanical turk faq.” [Online]. Available: https://www.mturk.com/worker/help

[37] “Acm sigplan empirical evaluation guidelines.” [Online]. Available: https://www.sigplan.org/Resources/EmpiricalEvaluation/

[38] P. M. Uesbeck, C. S. Peterson, B. Sharif, and A. Stefik, “A randomized controlled trial on the effects of embedded computer language switching,” in *ESEC/FSE ’20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 410–420. [Online]. Available: https://doi.org/10.1145/3368089.3409701

[39] “Osf.” [Online]. Available: osf.io/

[40] “Figshare.” [Online]. Available: https://figshare.com/

[41] “Zenodo.” [Online]. Available: https://zenodo.org/

[42] “Replication package for developer reading behavior while summarizing java methods: Size and context matters.” [Online]. Available: https://zenodo.org/record/2550768#.Yd3jIVjMI-Q

[43] N. J. Abid, B. Sharif, N. Dragan, H. Alrasheed, and J. I. Maletic, "Developer reading behavior while summarizing java methods: Size and context matters," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 2019, pp. 384–395.

[44] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*, 2018, pp. 286–28 610.

[45] B. Sharif, B. Clark, and J. I. Maletic, "Studying developer gaze to empower software engineering research and practice," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE 2016. New York, NY, USA: Association for Computing Machinery, 2016, p. 940–943. [Online]. Available: https://doi.org/10.1145/2950290.2983988