Trustworthiness Perceptions in Code Review: An Eye-tracking Study

Ian Bertram, Jack Hong, Yu Huang, Westley Weimer, Zohreh Sharafi
ianbtr,jackhong,yhhy,weimerw,zohrehsh@umich.edu
University of Michigan
Ann Arbor, Michigan

ABSTRACT

Background: Automated program repair and other bug-fixing approaches are gaining attention in the software engineering community. Automation shows promise in reducing bug fixing costs. However, many developers express reluctance about accepting machine-generated patches into their codebases.

Aims: To contribute to the scientific understanding and the empirical investigation of human trust and perception with regards to automation in software maintenance.

Method: We design and conduct an eye-tracking study investigating how developers perceive trust as a function of code provenance (*i.e.,* author or source). We systematically vary provenance while controlling for patch quality.

Results: In our study of ten participants, overall visual code scanning and the distribution of attention differed across identical code patches labeled as human- vs. machine-written. Participants looked more at the source code for human-labeled patches and looked more at tests for machine-labeled patches. Participants judged human-labeled patches to have better readability and coding style. However, participants were more comfortable giving a critical task to an automated program repair tool.

Conclusion: We find that there are significant differences in code review behavior based on trust as a function of patch provenance. Further, we find that important differences can be revealed by eye tracking. Our results may inform the subsequent design and analysis of automated repair techniques to increase developers' trust and, consequently, their deployment.

CCS CONCEPTS

• **Software and its engineering → Software creation and management**.

KEYWORDS

Trust, Code Review, Human factors, Automation, Eye tracking

ACM Reference Format:
Ian Bertram, Jack Hong, Yu Huang, Westley Weimer, Zohreh Sharafi. 2020. Trustworthiness Perceptions in Code Review: An Eye-tracking Study. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '20), October 8–9, 2020, Bari, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3382494.3422164

1 INTRODUCTION

Software maintenance, especially involving source code, is critical to software development. A number of source code management processes, tools and practices help developers coordinate as a team and improve software. *Code review* involves one or more developers examine a proposed change to a codebase (*e.g.,* a patch and its associated documentation) written by others and deciding whether the change should be accepted and integrated into the codebase or rejected for further refinement. Previous research has found that that code reviewers do not always prioritize the actual content and quality of a proposed patch, and studies of the effects of various biases (*e.g.,* gender bias) on code review have just begun [6, 14, 40]. This paper focuses on code review from a human trust perspective.

In Psychology, researchers have explored several metrics to infer and measure *trust.* Much previous work has focused on self-reporting (*i.e.,* subjective scales) such as think-aloud protocols, surveys, and interviews to measure trust [22], which suffer from the Hawthorn (observer) effect [4, 5] and may not be reliable in a software maintenance context [7, 15]. A few recent studies have used eye tracking as an objective, biologically-based measure to provide insights into the cognitive processes and the perception of trust in a continuous, non-subjective and non-intrusive manner [8, 9, 22]. These studies suggest a correlation between users' trust and their visual scanning behavior. By providing a dynamic pattern of visual attentions [16, 35], eye tracking offers numerous clues to underlying cognitive processes of the participants, helps to measure workload while performing tasks [29, 35, 36], and determines how and when participants choose and encode information [10, 29, 34].

This study assesses the perception of trust in code review using an eye tracker and examines whether the provenance of the patch impacts that perception of trust and any associated software engineering behaviors.

In the last decade, there has been a rapid rise in the research and usage of Automated Program Repair (APR) tools in both academia and industry [11, 27]. Recently, both larger (*e.g.,* Facebook's SapFix [24]) and smaller (*e.g.,* Janus Manager [12]) companies have deployed APR tools. However, many developers report an unwillingness to incorporate automated patches into their code bases [20] and expert programmers are less accepting of patches generated by APR tools [31]. However, little research to date has investigated APR from a human factors perspective focusing on the psychological processes behind the perception of trust (cf. [1, 2, 7, 28, 31]).

In this work, we present the preliminary results of a controlled human study involving ten participants to evaluate the potential bias that developers may have toward automated tools. Controlling for quality, we manipulate the provenance (*i.e.,* author or source) of a patch, labeling it as either human- or machine-written, regardless of its actual source. We measure code review outcomes and performance coupled with visual attention distributions to provide insights into developers' cognitive processes when evaluating and reviewing six code patches. By manipulating the apparent author name (*i.e.,* by labeling patches as human-generated vs. machine-generated), we perform a controlled experiment and obtain a lens to investigate participants' subjective trust and affected or biased actions, purely as a function of patch provenance, independent of patch quality.

Our results show that developers' perception of trust in the patch's author plays a crucial role in evaluating patches. While we observe no biases (measured by acceptance rate) or significant difference in high-level software maintenance outcomes (*e.g.,* accuracy, time spent, etc.), our recorded eye-movements data reveals different scanning patterns for human-labeled patches compared to the machine-labeled ones. For example, participants spend more visual effort evaluating the context class and its methods for human-labeled patches and spend more time evaluating the tests for machine-labeled patches, in a statistically-significant manner. Moreover, our participants report that human-labeled patches are of higher quality with regards to coding style and readability, but at the same time they are more likely to assign a critical task to an automated tool rather than a human developer. This positive attitude toward automated patches in a specific circumstance (*i.e.,* critical deployment) provides nuance to some previous work [31] that found a more generally negative attitude overall. APR patches often overfit to an available test suite and thus violate conventional programming approaches, can be complex or difficult to read, and may not address the root of the problem (*e.g.,* [19, 21]). Since we control for patch quality, our results highlight the importance of improving the trust perception of APR patches (*i.e.,* independent of their quality per se) to increase their use and acceptance.

2 RELATED WORK

In this section, we present background on eye tracking and trust studies in code review and automated program repair.

Trust and automated program repair. Automated program repair (APR) generates patches to fix defects in existing source code. While a rich body of research has been done on techniques, efficiency and quality concerns for APR (see [11, 27] for surveys), researchers have focused on trust in machine-generated repairs in recent years. In the past, research has investigated the human trust process in general (*i.e.,* not software engineering) automation, covering various aspects such as analyzing the effect of user personality over perceptions of x-ray screening tasks [26] and personal factors in ground collision avoidance software [23]. However, investigation on APR from human factors perspectives is relatively new. In [31], the authors found inexperienced programmers trust APR more than human patches. Fry *et al.* [7] found what humans report as being critical to patch maintainability might be different from what is actually more maintainable. Kim *et al.* [17] generated candidate patches following certain patterns and found that human developers view these pattern-based candidates as more acceptable, but did not compare acceptability against a control group of human-written patches for the same set of bugs.

Eye tracking and code review. Modern eye-tracking is an unobtrusive measure for visual focus by providing a reliable recording of eye gaze data [10, 29, 36]. Researchers have used eye-tracking techniques to investigate the viewing strategies of developers while performing code review tasks. These studies include analyzing the gaze patterns of developers in code review [41], understanding the impact of expertise in viewing strategies [37], and detecting suspicious code elements through viewing patterns [3]. They found that a complete scan of the whole code helps students to find defects f and share similar findings on code reading patterns. Besides general code reviewing patterns, Ford *et al.* used eye-tacking to study the influence of supplemental technical signals (such as the number of followers, activity, names, or gender) on Pull Request acceptance via an eye tracker [6].

To the best of our knowledge, this is the first eye-tracking study of examining trust toward patches of controlled quality labeled as generated by either machine or human developers.

3 EXPERIMENT SETUP AND METHOD

We recruited ten participants and conducted an exploratory study to investigate how developers trust and review source code on a large open-source project. Each participant worked on six code review tasks and reviewed six patches while we recorded their eye-movement data. All materials used in the study, along with de-identified responses, are available at the project's website.[1]

3.1 Experiment Measures

We assess both participant performance (*i.e.,* objective behavior during code review) and also trust intentions (*i.e.,* subjective self-evaluations).

Performance: we record the participant's performance on code review tasks in terms of the the amount of cognitive load (visual effort) measured by eye-tracking data, the amount of time spent finishing the task (total time spent), and the number of correct answers (accuracy).

Eye gaze data is typically divided into two categories [30]. First, a *fixation* is a spatially-stable eye gaze that lasts for approximately 200–300 ms. Researchers in psychology claim that most of the information acquisition and processing occur during fixations [16, 29] and that a small set of fixations suffices for the human brain to acquire and process a complex visual input [10, 16]. Second, a *saccade* is a continuous and rapid eye-gaze movement that occurs between fixations. Cognitive processing during saccades is minimal.

Eye gaze data is also studied with respect to certain *areas of interest* (AOIs) in a stimulus. AOIs are manually defined by the experimenter based on research questions and variables [10, 35]. In this experiment, we consider the following AOIs: 1) bug report text, 2) entire class file containing the patched code, 3) specific methods containing the patched code, 4) entire unit test file containing tests that evaluate the patch, and 5) specific unit tests methods that evaluate the patch.

[1]https://web.eecs.umich.edu/~weimerw/data/trust-perception/

Table 1: Eye tracking metrics used to evaluate participants' trust while reviewing human- vs. machine-labeled patches.

Metric	Description	Hypotheses tested in this paper
Average Fixation Duration (ms)	The average duration of all fixations in an AOI or the stimulus.	Lower trust leads to longer fixation durations [22].
Fixation Count	The number of attention shifts needed to finish the task [16].	Lower trust leads to a less organized search, which results in a higher number of fixations and less efficiency [22].
Average Saccade Length (px)	The average length of all the saccades in an AOI or the stimulus.	Lower trust leads to a more random, less efficient search, associated with shorter saccades.

Table 2: Participant demographics.

Characteristics	All (10)	Men (5)	Women (5)
Age (n (%))			
18-25	8 (80%)	3	5
25-30	2 (20%)	2	0
Class standing (n (%))			
2nd & 3rd year	3 (30%)	3	0
4th Year	2 (20%)	0	2
M.S./PhD	5 (50%)	2	3

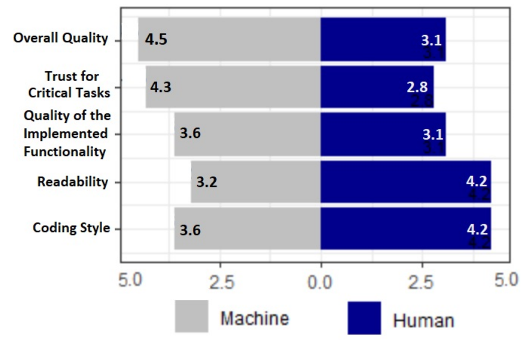

Figure 1: Participants' perception and evaluation of patches labeled as machine- or human-generated.

We consider three metrics (shown in Table 1) related to efficiency in searching and finding relevant data (*i.e.,* fixation count) and extracting information and an increased strain on the working memory (*i.e.,* fixation and saccade duration). We hypothesize that low trust leads to lower efficiency, manifested as longer fixations, a higher number of fixations, and shorter saccades.

Trust Intentions: we use subjective Likert-scale trust ratings, consisting of five questions about the overall quality of the patches, the level of trust for critical tasks, the quality of the implemented functionality, patch readability, and patch coding style.

3.2 Participants and Recruitment

In our IRB-approved study, we recruited ten undergraduate and graduate students in the Department of Computer Science at *institution elided for blind review.* Participants completed questionnaires to gather basic demographic information, reported in Table 2. Participants were recruited through email and were compensated with a $25 voucher for their participation.

We also asked participants about their experience and familiarity with programming, Java, and working with IDEs. Participants reported an average of 4.5 years (SD = 0.5) of programming experience. All participants were familiar with OOP and Java and had previous experience working with IDEs. Siegmund *et al.* [38] reported that self-estimation is a reliable way to judge programming experience, especially when working with students.

3.3 Software System and Tasks

We choose *jFreeChart* as the specimen system in this experiment. *jFreeChart* is a large open-source Java project that allows users to draw and display charts in their applications. We use version 1.1.0, released in 2015, which involves about 300 KLOC and about 94,000 Java classes. To present realistic code review tasks, we choose six historical bugs shown in Table 3. One-third of the patches were proposed by an APR tool and chosen from Defects4j-Repair project [25], while the rest were historical human repairs selected from the project's repository. Each patch consists of code samples ranging from 8 to 76 (Mean: 11 and SD: 13) lines of code and between 3 and 41 (Mean: 26 and SD: 23) repaired lines.

We randomly assigned the order of the six code review tasks for each participant. Following our experimental control, we randomly set the reported author label of three patches to be an automated algorithm (*i.e.,* "jGenProg") and the other three to be a particular invented human (*i.e.,* "David Gilbert"). Each task included evaluating a patch by examining a bug report file that contains the following information: actual bug report number and the date of the report, the priority (high or low), a summary, the number and the names of failing unit tests, author of the patch and the patch itself which shows the code differences between the old version and the proposed fix. During the experiment, participants had access to the whole code repository and could freely navigate it through the Eclipse IDE.

3.4 Procedure

Participants completed demographic information and background surveys online before the study. We conducted the experiment in a quiet room with an eye-tracking system; the participants were seated approximately 70 cm away from the screen in a comfortable swivel chair with armrests. Before running the experiment, all participants signed a consent form and the experimenter verbally explained the experiment's procedure in detail. Participants were informed that the experiment consisted of one Java project and that they would be sequentially reviewing six patches. The experimenter did not explain the particular goal of the experiment.

Participants were given ten minutes per code review task and were instructed to inform the experimenter if they finished early to stop the tracking. To mitigate any learning effects, participants received the six tasks in randomized order. To better control and avoid any other factors impacting the results, the participants were instructed to maintain the full-screen IDE setup, not use the debugger, and not to browse the internet.

Participants were instructed to mark each code review task (*i.e.,* each patch) as either "accept" and "reject" while evaluating the implemented functionality and quality of the patch. Participants also completed a post-questionnaire. To mitigate the stereotype threat [33], we asked questions about coding knowledge and experience at the end of the study. In particular, women and underrepresented minorities experience the negative stereotype that they have weaker abilities more strongly than others [39].

3.5 Equipment and Raw Measurements

We executed all experiments on a 27" monitor with a screen resolution of 1920x1080 pixels. We used the Tobii Pro X3-120 eye-tracker [13], which can locate eye-gaze data in a code document at a granularity of a single line of 10pt text. To support scrolling, switching between files, and editing files, we installed and used the iTrace 0.0.1A plugin [32], which gathers all necessary measurements while allowing participants to interact with source code and other artifacts naturally. We subjected the recorded raw data to an iTrace filter to generate fixation data.

4 DATA ANALYSIS AND EMERGING RESULTS

This research investigates developers' perceptions of trustworthiness in code patches and how patches's apparent provenance biases human efficiency and code review behavior. We hold the code quality constant and manipulate the patch's provenance by labeling the author as either human or machine. We focus the interpretation of our results around answers to the following research questions:

RQ1. How well do participants' self-reports regarding the role of patches' provenance align with our recorded data?

RQ2. How does a patch's provenance impact the participants' performance while reviewing the code?

RQ3. How does the provenance of a patch impact the participants' code review behavior while reviewing the code?

4.1 RQ1. Self-Reporting and Trust

All ten participants provided answers for the post-experiment questionnaire and evaluated the quality of code patches. Figure 1 summarizes the results. Regarding the readability and style, participants assigned a higher score to human-labeled patches in a statistically significant manner (Mann-Whitney test of readability and style scores combined, $W = 293$, $p = .008$).

When comparing the patches' overall quality, participants ranked machine-labeled patches higher, and the result is statistically significant (Mann-Whitney test, $W = 12$, $p = .003$). Participants also feel more comfortable assigning critical tasks to the machine rather than a human programmer (Mann-Whitney test, $W = 9$, $p = .001$).

In the same vein, Ryan *et al.* [31] reports that student programmers perceive machine-generated patch as more trustworthy. Previous research in automation [2, 31] reported a negative relation between tendency to trust and age. Our participants were young students with limited working experience, so this may have impacted the results. Further experimentation is required to study this relationship better.

4.2 RQ2. Performance Differences

We investigate whether the patch's provenance impacts the participants' overall performance measured by their accuracy, total time spent, and the visual effort (See Table 4). The test of proportion (Chi-squared test for significance) for accuracy and Wilcoxon signed-rank test for time shows no difference between human- vs. machine-labeled patches. On average, our participants are 8% less likely to accept human-labelled patches. Yet, Chi-squared test results in no significant differences. The visual effort is measured by fixation count, average fixation duration, and average saccade length. Longer fixation time and saccade indicate more considerable effort and higher cognitive load. No significant effect of provenance was found on eye-tracking metrics in isolation.

Previous work reports that participants' trust significantly impacts various eye-tracking metrics over the system [8, 9, 22] However, even though participants' self-report acknowledges a bias against both machines (readability) and humans (quality), we find no impact of the patch's provenance on participants' high-level performance. With regard to overall behavior, such as acceptance rates or time spent, participants were not influenced by provenance. We next investigate how that equal time was spent differently as a function of provenance.

4.3 RQ3. Differences in Code Review Behaviour

We consider the impact of provenance on developers' code review behavior by analyzing the pattern and distribution of visual attention. We calculate the total fixation time spent on each AOI and compare the distribution of attention across AOIs for machine-labeled vs. human-labeled patches. The general align-and-rank non-parametric factorial analysis [42] reveals that there is a significant interaction between provenance and the distribution of visual attention ($F(1, 5) = 2.149, p < 0.05$). As shown in Figure 2, the code scanning behavior of participants is different across patches. These results confirm that the relevance of different code areas varies significantly for participants based on the patch's apparent author. Although higher-level metrics (*e.g.,* total time) do not capture a difference, we observe that participants use different scanning behavior patterns and attention distributions while reviewing code.

Our participants spent more visual effort evaluating the class and its methods relevant to the bug while working on human-labeled patches in a statistically significant manner (Wilcox test: $W = 154, p < 0.05$). In contrast, they spent more effort on unit-test class and its relevant unit tests for machine-labeled patches. As discussed in Section 4.1, our participants ranked human-labeled patches as of lower quality with regards to the implemented functionality. This may partially explain why they spent more time evaluating the code of the patch for human-labeled ones compared to the machine ones. Analyzing unit tests is a standard approach for assessing the correctness of the patches. In the APR domain, unit tests play an even important role, as many APR-produced patches overfit to available test cases [19, 21]. Our participants

Table 3: Description of the patches, including a short summary of the reported bug along with impacted code elements.

		Scope of the solution			
ID	**Simplified Description**	Classes	Methods	Test Classes	Unit Tests
B1	Max-y value is not updated when copying a subset of TimeSeries.	1	1	1	1
B2	*XYSeries.addOrUpdate()* should add if duplicates are allowed.	1	1	1	1
B3	Potential Null Pointer Exception in *AbstractCategoryItemRenderer.getLegendItems()*	1	1	1	1
B4	If the label generator returns null, the PieChart must be created.	1	1	1	1
B5	After adding or removing items to the plots, the cached bounds must reset.	1	4	1	1
B6	We have a null pointer exception in *StatisticalBarRenderer* when one of a series in a category has no data.	1	1	1	4

Table 4: Pair-wise comparisons of performance metric using Chi-squared test for accuracy and acceptance rate and non-parametric Wilcoxon Test ($\alpha = 0.05$) for time and visual effort metrics. Significant results ($p < 0.05$) are bolded. We find no impact of the patch's provenance on participants' performance and acceptance rate.

	Mean (Std. Dev.)		
	Machine	Human	p
Acceptance rate	0.60	0.52	0.7
Accuracy	0.65	0.6	1
Time spent (s)	348.2 (177.6)	350.2 (141.4)	0.7
Avg Fix. duration (ms)	254.0 (82.9)	231.7 (83.6)	0.3
Fix. count	352.8 (224.0)	338.9 (238.1)	0.8
Avg sacc. length (px)	101.6 (13.6)	103.5 (15.3)	0.6

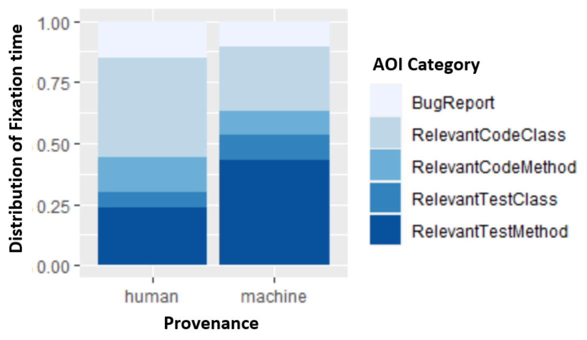

Figure 2: Distribution of fixation times across AOIs for human vs. machine patch labels, averaged for all participants. Participants put more attention on reading and processing the relevant class and method for human-labelled patches, but more time analyzing unit-test class and the relevant unit test for machine labels.

ranked machine-labeled code as less readable, and such a bias may prioritize inspection of tests instead. Either the desire to understand APR patch correctness or the relative perceived readability of the code may partially explain why participants spent more time on tests for machine-labeled patches.

5 THREATS TO VALIDITY

Several factors potentially affect the validity of our study. We use only one system, so its quality and complexity might influence the study. We mitigate this risk by choosing an open-source project, written in a popular programming language, and it is a reasonably large and complex project by general software engineering standards. Similarly, we select historic bugs from the project repository, so they are indicative of the real-world issues being reported.

We minimize the interaction between our team and participants to mitigate biases related to learning or using individual participants' identities. We did not inform the participants about the study's precise goals to alleviate hypothesis guessing and apprehension. We recruited a small number of participants, so we cannot consider the population large enough to generalize. Also, the majority of our participants are undergraduates. Yet, as evaluating the impact of expertise is not the goal of this study, using students as participants may be acceptable [18]. Finally, to account for conclusion validity, we choose well-documented eye-tracking metrics and analyses [34, 35].

6 CONCLUSION

We assess the perception of trust and the potential biases that developers may have toward automated tools in code review.

Participant self-reports show a bias against machines concerning code readability. Our participants also ranked machine-labeled patches as higher quality and reported higher trust in them for critical tasks. However, our empirical results no high-level performance differences between machine- and human-labeled patches. Also, we also find no biases against or in favor of automation while comparing the acceptance rate.

We find that the provenance influence the code review behavior of participants. Our attention distribution analysis reveals that the importance of different areas in the code changes based on perceived authorship. Our study sheds some light on developers' perception of trust and its impact on their behavior. It opens doors for further investigations that benefit various topics, including automated program repair and code reuse.

REFERENCES

[1] Gene M Alarcon, Rose Gamble, Sarah A Jessup, Charles Walter, Tyler J Ryan, David W Wood, and Chris S Calhoun. 2017. Application of the heuristic-systematic model to computer code trustworthiness: The influence of reputation and transparency. *Cogent Psychology* 4, 1 (2017), 1389640.

[2] Gene M. Alarcon and Tyler J. Ryan. 2018. Trustworthiness Perceptions of Computer Code: A Heuristic-Systematic Processing Model. In *Proceedings of the 51st Hawaii International Conference on System Sciences*.

[3] Andrew Begel and Hana Vrzakova. 2018. Eye Movements in Code Review. In *Proceedings of the Workshop on Eye Movements in Programming*. Article 5, 5 pages. https://doi.org/10.1145/3216723.3216727

[4] Anneli Eteläpelto. 1993. Metacognition and the expertise of computer program comprehension. *Scandinavian Journal of Educational Research* 37, 3 (1993), 243–254.

[5] Quyin Fan. 2010. *The Effects of Beacons, Comments, and Tasks on Program Comprehension Process in Software Maintenance*. Ph.D. Dissertation. University of Maryland, Baltimore County, Catonsville, MD, USA. Advisor(s) Norcio, Anthony F. AAI3422807.

[6] Denae Ford, Mahnaz Behroozi, Alexander Serebrenik, and Chris Parnin. 2019. Beyond the code itself: how programmers really look at pull requests. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Society (ICSE-SEIS)*. IEEE, 51–60.

[7] Zachary P. Fry, Bryan Landau, and Westley Weimer. 2012. A human study of patch maintainability. In *International Symposium on Software Testing and Analysis, ISSTA 2012, Minneapolis, MN, USA, July 15-20, 2012*, Mats Per Erik Heimdahl and Zhendong Su (Eds.). ACM, 177–187. https://doi.org/10.1145/2338965.2336775

[8] Claudia Geitner, Ben D Sawyer, S Birrell, P Jennings, L Skyrypchuk, Bruce Mehler, and Bryan Reimer. 2017. A link between trust in technology and glance allocation in on-road driving. (2017).

[9] Christian Gold, Moritz Körber, Christoph Hohenberger, David Lechner, and Klaus Bengler. 2015. Trust in automation–Before and after the experience of take-over scenarios in a highly automated vehicle. *Procedia Manufacturing* 3 (2015), 3025–3032.

[10] Joseph H. Goldberg and Jonathan I. Helfman. 2010. Comparing Information Graphics: A Critical Look at Eye Tracking. In *Proceedings of the 3rd BEyond Time and Errors: Novel evaLuation Methods for Information Visualization Workshop* (Atlanta, Georgia) *(BELIV '10)*. ACM, New York, NY, USA, 71–78. https://doi.org/10.1145/2110192.2110203

[11] Claire Goues, Stephanie Forrest, and Westley Weimer. 2013. Current Challenges in Automatic Software Repair. *Software Quality Journal* 21, 3 (Sept. 2013), 421–443. https://doi.org/10.1007/s11219-013-9208-0

[12] Saemundur O. Haraldsson, John R. Woodward, Alexander E. I. Brownlee, and Kristin Siggeirsdottir. 2017. *Fixing Bugs in Your Sleep: How Genetic Improvement Became an Overnight Success.* https://doi-org.proxy.lib.umich.edu/10.1145/3067695.3082517

[13] https://www.tobiipro.com/. 2001. Online; Accessed 17-07-2020.

[14] Yu Huang, Kevin Leach, Zohreh Sharafi, Nicholas McKay, Tyler Santander, and Westley Weimer. 2020. Investigating Gender Bias and Differences in Code Review: Using Medical Imaging and Eye-Tracking. In *International Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM/SIGSOFT.

[15] Yu Huang, Xinyu Liu, Ryan Krueger, Tyler Santander, Xiaosu Hu, Kevin Leach, and Westley Weimer. 2019. Distilling Neural Representations of Data Structure Manipulation Using FMRI and FNIRS. In *Proceedings of the 41st International Conference on Software Engineering (ICSE '19)*. IEEE Press, Montreal, Quebec, Canada, 396–407. https://doi.org/10.1109/ICSE.2019.00053

[16] Marcel A Just and Patricia A Carpenter. 1980. A theory of reading: from eye fixations to comprehension. *Psychological review* 87, 4 (1980), 329.

[17] Dongsun Kim, Jaechang Nam, Jaewoo Song, and Sunghun Kim. 2013. Automatic patch generation learned from human-written patches. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 802–811.

[18] Barbara A. Kitchenham, Shari Lawrence Pfleeger, Lesley M. Pickard, Peter W. Jones, David C. Hoaglin, Khaled El Emam, and Jarrett Rosenberg. 2002. Preliminary Guidelines for Empirical Research in Software Engineering. *IEEE Transactions on Software Engineering* 28, 8 (Aug. 2002), 721–734. https://doi.org/10.1109/TSE.2002.1027796

[19] Xuan-Bach D. Le, Ferdian Thung, David Lo, and Claire Le Goues. 2018. Overfitting in semantics-based automated program repair. *Empirical Software Engineering* 23, 5 (2018), 3007–3033. https://doi.org/10.1007/s10664-017-9577-2

[20] Claire Le Goues, Michael Dewey-Vogt, Stephanie Forrest, and Westley Weimer. 2012. A Systematic Study of Automated Program Repair: Fixing 55 out of 105 Bugs for $8 Each. In *International Conference on Software Engineering*.

[21] Fan Long and Martin Rinard. 2016. Automatic Patch Generation by Learning Correct Code. In *Principles of Programming Languages*. https://doi.org/10.1145/2837614.2837617

[22] Y. Lu and N. Sarter. 2019. Eye Tracking: A Process-Oriented Method for Inferring Trust in Automation as a Function of Priming and System Reliability. *IEEE Transactions on Human-Machine Systems* 49, 6 (Dec 2019), 560–568. https://doi.org/10.1109/THMS.2019.2930980

[23] Joseph B Lyons, Nhut T Ho, William E Fergueson, Garrett G Sadler, Samantha D Cals, Casey E Richardson, and Mark A Wilkins. 2016. Trust of an automatic ground collision avoidance technology: A fighter pilot perspective. *Military Psychology* 28, 4 (2016), 271–277.

[24] A. Marginean, J. Bader, S. Chandra, M. Harman, Y. Jia, K. Mao, A. Mols, and A. Scott. 2019. SapFix: Automated End-to-End Repair at Scale. In *International Conference on Software Engineering: Software Engineering in Practice*. 269–278.

[25] Matias Martinez, Thomas Durieux, Romain Sommerard, Jifeng Xuan, and Martin Monperrus. 2016. Automatic Repair of Real Bugs in Java: A Large-Scale Experiment on the Defects4J Dataset. *Springer Empirical Software Engineering* (2016). https://doi.org/10.1007/s10664-016-9470-4

[26] Stephanie Merritt, Lei Shirase, and Garett Foster. 2020. Normed Images for X-ray Screening Vigilance Tasks. *Journal of Open Psychology Data* 8, 1 (2020).

[27] Martin Monperrus. 2018. Automatic Software Repair: A Bibliography. *ACM Comput. Surv.* 51, 1, Article 17 (Jan. 2018), 24 pages. https://doi.org/10.1145/3105906

[28] Martin Monperrus, Simon Urli, Thomas Durieux, Matias Martinez, Benoit Baudry, and Lionel Seinturier. 2019. Repairnator Patches Programs Automatically. *Ubiquity* 2019, July, Article 2 (July 2019), 12 pages. https://doi.org/10.1145/3349589

[29] Alex Poole and Linden J. Ball. 2005. Eye Tracking in Human-Computer Interaction and Usability Research: Current Status and Future. In *Prospects", Chapter in C. Ghaoui (Ed.): Encyclopedia of Human-Computer Interaction. Pennsylvania: Idea Group, Inc.* Information Science Reference, Hershey, PA, 1–5.

[30] K. Rayner. 1978. Eye movements in reading and information processing. *Psychological Bulletin* 85, 3 (1978), 618–660.

[31] Tyler J. Ryan, Gene M. Alarcon, Charles Walter, Rose F. Gamble, Sarah A. Jessup, August A. Capiola, and Marc D. Pfahler. 2019. Trust in Automated Software Repair - The Effects of Repair Source, Transparency, and Programmer Experience on Perceived Trustworthiness and Trust. In *Proceedings of Cybersecurity, Privacy and Trust - First International Conference, HCI-CPT 2019 Orlando, FL, USA, July (Lecture Notes in Computer Science)*, Abbas Moallem (Ed.), Vol. 11594. Springer, 452–470. https://doi.org/10.1007/978-3-030-22351-9_31

[32] Timothy R. Shaffer, Jenna L. Wise, Braden M. Walters, Sebastian C. Müller, Michael Falcone, and Bonita Sharif. 2015. iTrace: Enabling Eye Tracking on Software Artifacts Within the IDE to Support Software Engineering Tasks. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering* (Bergamo, Italy) *(ESEC/FSE 2015)*. 954–957.

[33] Jenessa R Shapiro and Steven L Neuberg. 2007. From stereotype threat to stereotype threats: Implications of a multi-threat framework for causes, moderators, mediators, consequences, and interventions. *Personality and Social Psychology Review* 11, 2 (2007), 107–130.

[34] Zohreh Sharafi, Timothy Shaffer, Bonita Sharif, and Yann-Gaël Guéhéneuc. 2015. Eye-tracking metrics in software engineering. In *Proceeding of 2015 Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 96–103.

[35] 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* (2020), 1–47.

[36] 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.

[37] 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 *Symposium on Eye Tracking Research and Applications*. https://doi.org/10.1145/2168556.2168642

[38] Janet Siegmund, Christian Kästner, Jörg Liebig, Sven Apel, and Stefan Hanenberg. 2014. Measuring and modeling programming experience. *Empirical Software Engineering* 19, 5 (2014), 1299–1334.

[39] Claude M Steele and Joshua Aronson. 1995. Stereotype threat and the intellectual test performance of African Americans. *Journal of personality and social psychology* 69, 5 (1995), 797.

[40] Josh Terrell, Andrew Kofink, Justin Middleton, Clarissa Rainear, Emerson R. Murphy-Hill, Chris Parnin, and Jon Stallings. 2017. Gender differences and bias in open source: pull request acceptance of women versus men. *PeerJ Comput. Sci.* 3 (2017), e111. https://doi.org/10.7717/peerj-cs.111

[41] Hidetake Uwano, Masahide Nakamura, Akito Monden, and Ken-ichi Matsumoto. 2006. Analyzing Individual Performance of Source Code Review Using Reviewers' Eye Movement. In *Eye Tracking Research Applications*. 133–140.

[42] Jacob O Wobbrock, Leah Findlater, Darren Gergle, and James J Higgins. 2011. The aligned rank transform for nonparametric factorial analyses using only anova procedures. In *Proceedings of the SIGCHI conference on human factors in computing systems*. ACM, 143–146.