Towards debiasing code review support

Jetzen, Tobias; Devroey, Xavier; Matton, Nicolas; Vanderose, Benoît

DOI:
10.48550/arxiv.2407.01407

Publication date:
2024

Towards debiasing code review support

Tobias Jetzen
NADI, University of Namur
Namur, Belgium

Xavier Devroey
NADI, University of Namur
Namur, Belgium
xavier.devroey@unamur.be

Nicolas Matton
NADI, University of Namur
Namur, Belgium
nicolas.matton@unamur.be

Benoît Vanderose
NADI, University of Namur
Namur, Belgium
benoit.vanderose@unamur.be

***Abstract*—Cognitive biases appear during code review. They significantly impact the creation of feedback and how it is interpreted by developers. These biases can lead to illogical reasoning and decision-making, violating one of the main hypotheses supporting code review: developers' accurate and objective code evaluation. This paper explores harmful cases caused by cognitive biases during code review and potential solutions to avoid such cases or mitigate their effects. In particular, we design several prototypes covering confirmation bias and decision fatigue. We rely on a developer-centered design approach by conducting usability tests and validating the prototype with a user experience questionnaire (UEQ) and participants' feedback. We show that some techniques could be implemented in existing code review tools as they are well accepted by reviewers and help prevent behavior detrimental to code review. This work provides a solid first approach to treating cognitive bias in code review.**

***Index Terms*—cognitive bias, code review, user-centered design**

I. Introduction

One of the many software development activities taken to ensure code quality is *code review*. Code review consists of methodical code assessments that follow pre-defined guidelines and are supported by various tools to identify potential bugs, increase code readability and understandability, help developers learn the source code (i.e., code knowledge transfer), etc. Practically, code review is performed by a developer (i.e., a *reviewer*), usually other than the *author* of the code being reviewed. This reviewer, potentially helped by various tools, will read the code and make comments, ask questions, and request changes in the code that the author will take care of. Once the author and the reviewer are satisfied, the code is included (i.e., *merged*) into the code base. The main goals of code review are to prevent defects, enable knowledge, check the code readability, enforce maintainability standards, etc., on recently modified source code [3]. Code reviews also serve as gatekeepers to prevent developers from committing arbitrary code without verification [33].

Like many software engineering activities, code review not only involves applying technical knowledge but heavily relies on social interactions between the reviewer and the author of the code [8], [12], [27], [34]. Such social-based activities are heavily influenced by cognitive and social aspects often neglected. This research focuses on *cognitive biases* [2], and more specifically, their triggers and potential effects on code review quality. Our first goal is to identify factors triggering cognitive biases for the reviewer or the author. Our second goal is to design solutions to avoid such biases or mitigate their effects.

In this short paper, we design solutions addressing the triggers of *confirmation bias* and the effects of *decision fatigue*. For that, we follow a user-centered (here, developer-centered) design approach [26], [28], [29]. In the first phase, we explore potentially harmful situations and design theoretical solutions to prevent or mitigate biases. In the second phase, we aim to improve the designed solutions by conducting usability tests with beginners using a prototype based on an existing code review tool. These tests serve as feedback to gather the users' requirements for an acceptable solution. To achieve this, we iterate multiple times over the prototype. Finally, we conduct an evaluation of the prototype's final result by testing the user experience with the standard *User Experience Questionnaire* (UEQ) [36], [37]. With the prototype as a final result and an evaluation of its usability (see our replication package [1]), we propose a first work on solving problematic relationships between cognitive bias and code review. The prototype will serve as the basis for developing a functional tool to evaluate the impact of our solutions on confirmation bias and decision fatigue.

II. Background and related work

In psychology, cognitive biases (denoted biases hereafter) refer to instances where human cognition consistently generates representations that are systematically distorted when compared to objective reality [18]. Unlike logical fallacies, which are arguments based on invalid reasoning, biases are patterns of thinking that affect how we interpret new information and processes. Biases are applied systematically and influence our behavior, opinions, and decisions. Causes of biases are rooted in heuristics [2], [18], [20], [21], [42]: shortcuts or rules of thumb used by our brain to solve a problem or judge a situation quickly. For instance, people with a higher social position tend to apply stereotypical views on others more often than those with a more precarious position, who invest more time and energy in social judgment [18]. However, in general, precisely identifying the exact causes of a specific bias is challenging [27].

To eliminate biases (i.e., *debiasing*), previous research has shown that neither applying more effort nor being more experienced in a field helps mitigate cognitive biases [14]. However, training on cognitive biases and applying specific techniques can make a substantial difference. This has been proven not only for experts in a field but also to affect the judgment of non-experts [9].

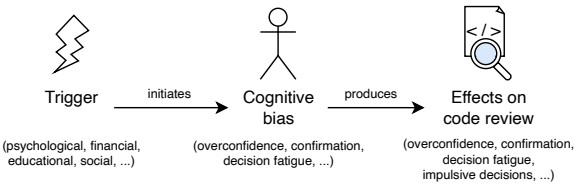


Fig. 1. Relationship between triggers, cognitive biases, and their effects.

Practically, as illustrated in Figure 1, when we know a cognitive bias, we can identify **triggers** initiating it and what **impact** the bias has on the investigated activity. In this work, a trigger is considered a specific environmental condition, enabling a cognitive bias. One cognitive bias can potentially be triggered by multiple elements [32]. Once triggered, it affects the person's activity. In our case, code reviews can be impacted severely by cognitive biases. One cognitive bias, in turn, can produce multiple effects during the different activities.

Software engineering being a process heavily relying on human effort and involvement, cognitive biases have also been explored for various software engineering activities [8], [12], [15], [27], [34]. For instance, Barroso et al. [4] investigated developers' personalities' influence on their tasks. They have shown that the quality of the product depends on the interaction between members of the team combined with their professional capabilities and, therefore, on the interpretation and reactions to feedback. Going further, Spadini et al. [40] investigated the effect of existing review comments on code review and showed that reviewers are subject to availability bias when performing reviews. More recent research has evidenced the importance of appropriate techniques to deal with cognitive biases during code review, such as checklists to potentially lower developers' cognitive load [16]; avoiding destructive criticisms not to decrease motivation [17]; or guideline to deal with confusion during code reviews [11].

In a recent study, Fagerholm et al. [12] have identified several future research directions for cognition in software engineering, including perception and software quality, which have still received very little attention. In this work, we focus on debiasing code review to avoid confirmation bias (i.e., *triggers*) and mitigating potential *effects* of decision fatigue.

a) Confirmation bias: One of the most researched cognitive biases in psychology is the confirmation bias. When talking about confirmation bias, one refers to the collection, interpretation, analysis and research for information in a way that confirms one's prior beliefs instead of searching for information disproving them [19], [31]. In practice, once the mind adopts an opinion, it does everything to support it, leading to wrong decisions defying the sense of logical reasoning. For instance, the positive test bias leads developers to test only to confirm the code, instead of disproving it [34], [41]. Tests are more effective with data, which is designed to disconfirm hypothesis [25]. In general, and not only during tests, one's goal should be to fail the code in order to reduce defect density [7].

b) Decision fatigue: A high number of decisions to make, each requiring to process information, over a short period of time leads to the depletion of internal resources, also referred to as *ego depletion* [5]. When ego depletion manifests as decision fatigue, it causes attentional deficit, impulsive decisions, and leads to postponing decisions with the intention to look at them later [10]. Finally, people subjected to decision fatigue tend to have an impaired ability to make trade-offs; they prefer acting in a passive role and make irrational judgments. Unfortunately, such changes in behavior are hard to recognize [30].

TABLE I
SCENARIOS FOR CONFIRMATION BIAS: TRIGGERS, EFFECTS, AND REMEDIES

Trigger:	The developer gets low-quality feedback, hurting their self-esteem.
Effect:	The developer refuses recommendations from the feedback to protect their self-esteem.
Remedies:	*Constructive feedback*. Prevent the bias by providing the reviewer with advice about how to give constructive feedback. *Review feedback*. Prevent the bias by suggesting to the reviewer to ask another developer for feedback about their review.
Trigger:	The reviewer is under time-pressure due to circumstances.
Effect:	The reviewer tries to validate the existing code instead of analyzing it objectively.
Remedy:	*Encourage brainstorming*. Mitigate the impact by providing a form with empty solution fields to encourage the reviewer to think about multiple solutions.

III. DEBIASING CODE REVIEW

Our approach addresses the triggers and effects of confirmation bias and decision fatigue. As mentioned in Section II, precisely identifying the exact causes and effects of a specific bias is challenging. In our first step, we define scenarios in which triggers and effects can clearly be identified and reproduced. Exploring more complex triggers and effects of different biases is part of our future work.

a) Confirmation bias: Research in psychology investigated confirmation bias a lot [7], [22], [34], providing a solid basis to research its relations to software engineering, especially to modern code review. We focus on two scenarios described in Table I, with potential solutions. The assumption is that how a reviewer builds the feedback influences the developer's perception and, therefore, their acceptance of the feedback (first line in Table I). Also, when reviewers see code changes, they are exposed to code that influences their perception during review [40]. Our assumption is that under time pressure, this phenomenon becomes more pronounced: a reviewer tends to search for fast review approval instead of correct implementation (second line in Table I).

b) Decision fatigue: Many triggers can initiate decision fatigue [5], [39]. We focus on the scenarios described in Table II. The first and second scenarios consider that a reviewer needs motivation to tackle new code or potentially new topics. Humans tend to perform small tasks where they are rewarded early: this behavior is called hyperbolic discounting [23]. However, during code review, a reviewer may get assigned a great number of reviews to do or review code requiring specific knowledge. Our assumption is that when decision fatigue is triggered due to circumstances that are unfavorable for starting tasks intensive in cognitive resources, the reviewer tends to procrastinate. In the third scenario, decision fatigue leads the

TABLE II
SCENARIOS FOR DECISION FATIGUE: TRIGGERS, EFFECTS, AND REMEDIES

Trigger:	The reviewer is over-solicited.
Effect:	The reviewer misses motivation to do reviews and postpones them for later (i.e., procrastination).
Remedies:	*Scheduled reviews*. Prevent the bias by limiting the amount of reviews to a maximum number and a calendar to schedule. *Observe needed time*. Prevent the bias by reminding the reviewer to halt when too much time is needed for review.
Trigger:	The reviewer misses knowledge about a specific topic in the code.
Effect:	The reviewer misses motivation to do reviews and postpones them for later (i.e., procrastination).
Remedy:	*Find an expert*. Prevent the bias by assigning the best fitting reviewer according to their experience in the topic.
Trigger:	The reviewer is working at times of day known for decreased internal resources (e.g., the end of the working day or after lunch).
Effect:	The reviewer makes impulsive comments instead of constructive suggestions for the author.
Remedy:	*Guide with comments*. Mitigate the impact by guiding the reviewer through the files with comments made by the author.
Trigger:	The reviewer lacks of experience in making code reviews.
Effect:	The reviewer skips code changes or parts of the code, leading to a lower understanding of the code.
Remedy:	*Help commenting*. Mitigate the impact by providing a form with keywords to help the reviewer to include all essential elements.

reviewer to make impulsive comments. Our assumption is that the comments will be expressed in a familiar way, leading to destructive feedback instead of constructive one. In the last scenario, our assumption is that understanding the code is key to making constructive comments for the author. A review under the effect of decision fatigue (i.e., not taking all the elements into account) might provoke misleading results in the feedback for the author.

Mitigating the impact of biases on code review requires dealing with either the trigger initiating the bias or its effects. Depending on the scenario, the former or the latter might be better suited.

IV. DESIGN OF THE CODE REVIEW SUPPORT

In the following section, we will design and test potential solutions to address the remedies described in Tables I and II. For that, we follow a user-centered (here, reviewer/developer-centered) design process [26], [28], [29] to guide the building of our support following an iterative approach. Figure 2 provides an overview of the process: due to time constraints, we limited the development to two iterations (i.e., *usability tests*), concluded by a *user experience test* of the designed solutions, performed with a different group of users than the one involved in the first and second usability tests. The tested solutions are developed as HTML prototypes. We relied on two groups of users: three for the first and second usability tests and five for the final user experience test. We employ a total of eight participants, which, according to Faulkner [13], helps identify, on average, more than 80% of the problems.

A. First usability test

First, we create an initial prototype based on the web tool Gerrit[1], similar to what can be found on GitHub. Then we

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

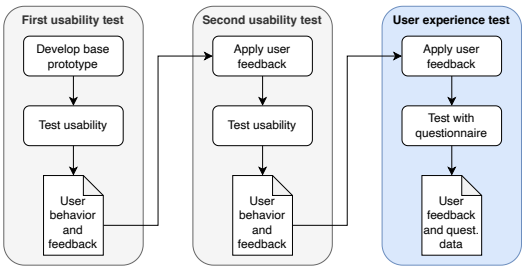


Fig. 2. Design, prototyping, and evaluation

implement the remedies listed in Tables I and II using the following techniques. All the prototypes can be found in our replication package [1].

a) Technique 1.1 — Advice: The goal is to help the reviewer create constructive feedback [17]. The first technique is based on advice displayed in the form of a list. This list can be opened by clicking the button *I need advice*. Following this action, a popup shows a list containing the advice. The advice originates from literature investigating how to achieve constructive feedback [6], [43].

b) Technique 1.2 — Form: Because under decision fatigue reviewers tend to make incomplete comments [16], the idea of a pre-structured form seems an appropriate choice. This way, the reviewer does not have to think about how to structure the comment. It should include the identification of a problem, a justification regarding why the discovery is considered a problem and also a suggestion to solve it. Here, three empty fields are available to encourage the reviewer to brainstorm multiple suggestions, thereby promoting more thorough and considered feedback.

c) Technique 1.3 -– Guide: To avoid the user skipping changes or even entire files from being reviewed, a guide is offered just before starting the review [11]. This guide consists of a certain amount of comments written by the author. They contain an explanation of why a certain change was made. When launching the guide, the reviewer's attention is immediately drawn to the first comment, surrounded by a red border, as decided by the author. When the reviewer decides that they understand the change, they click on the button *Next* to go to the next comment. Once having been through every step, the reviewer starts the actual review.

The test analyses the navigation through the prototype, usage of the tool, and reaction to the tool by the participants who act as developers or as reviewers, depending on the technique. Due to space constraints., we only report a summary of our observations in Table III. Detailed protocol and results are available in our replication package [1].

B. Second usability test

We improved the prototype following the observed behavior and the participants' suggestions. Some techniques get improved, two new techniques are added, and one is removed.

a) Technique 2.1 — Advice: In the current iteration, the advice popup was transformed into a drop-down list that is immediately visible when the comment tool is opened. This

TABLE III
SUMMARY OF THE USABILITY TESTS OBSERVATIONS

Usability test 1	
Technique 1.1 Advice	⊕ Most users apply the advice after reading it. ⊕ The short formulated advice is appreciated. ⊖ The popup button is not always noticed. ⊖ Not everybody wants to be advised. ⊖ Background color confuses participants.
Technique 1.2 Form	⊕ All fields get filled out. ⊕ The form provides a coherent structure. ⊖ Only one solution is given. ⊖ Some feel overwhelmed.
Technique 1.3 Guide	⊕ Everyone uses the guide. ⊕ No code change is skipped. ⊖ The guide could bias the reviewer's comment. ⊖ The `Next` button is not intuitive.
Usability test 2	
Technique 2.1 Advice	⊕ The advice is noticed and read immediately. ⊕ The advice impacts the overall comment. ⊖ Green color signifies already complete. ⊖ Participants mistake inciting items for to-do items.
Technique 2.2 Example	⊕ All participants use the technique. ⊕ Saves time to think about structure. ⊕ Comment analysis could be automated. ⊖ Participants type the keywords manually.
Technique 2.3 Quick search	⊕ All participants use the technique. ⊖ Some only use the quick search, without commenting.
Technique 2.4 Expert feedback	⊖ Most participants do not use the technique. ⊖ Most users are concerned about annoying colleagues.
Technique 2.5 Guide	⊕ Launch button is noticed faster. ⊕ Understanding the Next button is intuitive. ⊕ The Next button acts as an obligation to comment. ⊖ Unable to make comments inside the guide.

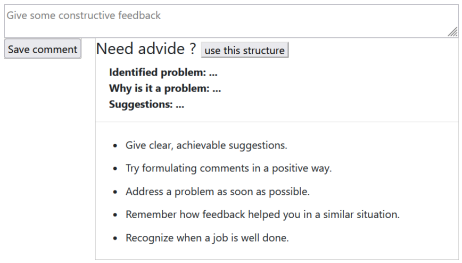
change ensures that every participant notices and reads the advice, unlike in previous iterations. As stated in Table III, the green background does not show positive effects. Also, though most elements can be used like in a checklist, some of them can not, because they are intended to incite the reviewer to analyze the code from another perspective.

b) Technique 2.2 — Example: In the previous iteration, a new technique emerged from the advice technique 1.1. Here, the technique uses an example, and participants follow the structure presented in the example. The feedback confirms appreciation, as shown in Table III.

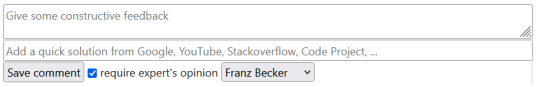
c) Technique 2.3 — Quick search: As requested in the first iteration, tasks should take less effort (i.e., fewer intermediary clicks). The form used before is now replaced with a quick search. It was used in different ways: first commenting, then searching for code snippets related to the comment, or searching for a solution before commenting. In the latter case, however, the comment only refers to the selected solution from the search without further explanation.

d) Technique 2.4 — Expert feedback: As for the previous technique, another help was requested: providing expert feedback allows for countering the effects of decision fatigue when the reviewer is prone to take a passive role. However, as observed during the tests, most participants do not use the expert feature because they feel they are experienced enough or do not want to annoy a senior colleague.

e) Technique 2.5 — Guide: In this iteration, the launch button for the guide is bigger and thus better visible, and



(a) Combination of advice and example.



(b) Combination of quick search and expert feedback.

Fig. 3. Combinations of techniques.

the button used to get to the next step is placed in a more understandable location. As stated in Table III, it has a positive impact. However, all participants complain about not being able to give comments while using the guide.

C. Final prototype

During the second usability test, multiple participants suggested combining techniques.

a) Technique 3.1 – Advice: Techniques 2.1 and 2.2 are combined to create constructive feedback, as displayed in Figure 3a. A new button is added to allow preparing the comment structure faster: it copies the key phrases "Identified problem", "Why is it a problem" and "Suggestions" into the comment field. The advice is located beneath the example.

b) Technique 3.2 – Assistant: Techniques 2.3 and 2.4 are combined to mitigate decision fatigue's effects (Figure 3b).

c) Technique 3.3 – Guide: Addressing the negative point of Technique 2.5 (⊖ in Table III) would not support the intended effect. Being able to comment continuously hinders the reviewer from following the guide to understand the code as a whole and not only partially. Therefore, Technique 3.3 remains equivalent to Technique 2.5.

V. USER EXPERIENCE TEST

The goal of this preliminary evaluation is to gain feedback about the user interface and its usability, not about the psychological aspects. Those aspects are left for future work.

An evaluation proceeds with each of the five participants, all experienced developers, familiar with code review, as follows: (i) we ask the participant about their experience in code review; (ii) depending on the answer, we explain and demonstrate what code reviews are and how tools can be used to assist the code review process; (iii) for each technique (Technique 3.1-3.3), we explain to the participant about the task (one per technique) to perform and proceed with the test. The explanation however takes place without mentioning the investigated aspects concerning cognitive biases. This is necessary to not bias the participant's opinion. (iv) after completing each task, the participant is asked to fill out a standard User Experience Questionnaire (UEQ) [24], [35], [36], [38]. The

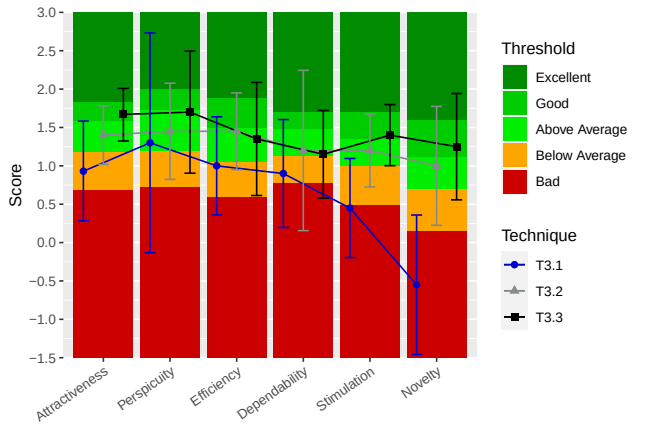


Fig. 4. Results of the UEQ benchmark for the advice with example (T3.1), the quick search with expert feedback (T3.2), and the guide (T3.3). Dots denote mean values, and error bars indicate 95% confidence intervals.

UEQ is a heavily validated state-of-the-art questionnaire measuring user experience following the predefined scales: Attractiveness (do users like or dislike the product), Perspicuity (is it easy to get familiar with the product?), Efficiency (can users solve their tasks without unnecessary effort?), Dependability (does the user feel in control of the interaction?), Stimulation (is it exciting and motivating to use the product?), and Novelty (is the product innovative and creative?). Additionally, we ask every participant after the questionnaire for personal feedback about the tested technique

We analyze questionnaire responses for each technique to evaluate if it improves user experience using the UEQ Data Analysis Tool [35]. This tool provides a quantitative analysis, converting responses from a scale of 1 to 7 to a scale of -3 (most negative) to +3 (most positive). It classifies the scale values into five categories, from excellent to bad, based on a benchmark of user interfaces [37]. Results for the three techniques are reported in Figure 4. Given the preliminary nature of the evaluation, the small sample size results in a less accurate quantitative analysis (as confirmed by the sometimes large confidence intervals in Figure 4), yet it still indicates trends in user experience, supported by the additional qualitative feedback (not reported in this short paper due to space constraints). The complete data are available in our replication package [1].

The UEQ results indicate that users perceive *Technique 3.1 - Advice* as understandable and easy to learn. Users rated the technique poorly on the stimulation scale, indicating it was neither exciting nor motivating, and perceived it as conventional. Users also find *Technique 3.2 - Assistant* interesting, exciting, and motivating, as well as efficient and easy to use, aligning with the technique's goals. The technique is seen as innovative in code reviews. Overall, the user experience results are positive, reflecting the feedback, but responses focus more on the *quick search* feature than the *expert feedback*. Finally, users find *Technique 3.3 - Guide* highly attractive and interesting, with clear indications of it being stimulating and innovative. It also receives positive evaluations for efficiency, with users feeling it meets expectations by providing support.

VI. Discussion and Future Work

A. Confirmation bias

Our base assumption was that non-constructive feedback during code review provokes confirmation bias. Our preliminary results from applying *Technique 3.1 - Advice* show that reviewers prefer guided, pre-defined structured comments but are generally unmotivated to follow written advice. Overall advice and examples are used quickly, without spending more than a few seconds to integrate them in the comments. Reviewers are willing to incorporate examples into the review process, with most agreeing on the positive effects of this technique. Feedback from our preliminary evaluation suggests that the technique helps prevent confirmation bias during code review. While this research does not quantitatively measure the extent to which the technique prevents confirmation bias, it includes a prototype solution and tests user experience. Further investigation with a larger sample size is necessary for representative quantitative results.

B. Decision fatigue

a) Incomplete comments: Our base hypothesis suggested that decision fatigue during code review leads to incomplete comments. From this, we proposed *Technique 3.2 - Assistant*. User experience tests show that the search tool encourages adding code snippets to comments and is well-received once users understand its purpose. This technique shows significant potential to mitigate decision fatigue. Conversely, the expert feedback tool is seen as annoying for senior colleagues, with reviewers avoiding it due to confidence in their comments. Currently, the expert feedback prototype cannot mitigate decision fatigue but could be improved through design adjustments. Overall, the design and layout significantly influence the effectiveness of the tools. User understanding and willingness to use a technique depend heavily on the interface design, suggesting that alternative designs might yield better results.

b) Skipping code changes: Our hypothesis also supports that decision fatigue during code review leads to skipping short, large, or complex changes. *Technique 3.3 - Guide* can prevent this by helping reviewers address important changes individually. User experience tests show that reviewers consistently use the guide, following it to the end without skipping any changes. Participants appreciated the guide, indicating it effectively prevents decision fatigue from causing skipped reviews. Thus, the technique helps mitigate certain effects of decision fatigue during code review.

C. Future work

The limited participant number constrained our quantitative data, and the high variance in some of the responses calls for further investigations. However, these results still helped identify tendencies and refine prototypes. Our future research will focus on design aspects for better outcomes, further prototype development, and extending the scope to other cognitive biases. User feedback also suggests automating review tasks

and providing context-sensitive feedback. Ultimately, these prototypes could evolve into fully functional tools for real-world application.

REFERENCES

[1] A. Anonymous, "Towards debiasing code review support replication package," https://anonymous.4open.science/r/psychopass-F3C7/.

[2] H. R. Arkes, "Costs and benefits of judgment errors: Implications for debiasing," *Psychological Bulletin*, vol. 110, no. 3, pp. 486–498, 1991, place: US Publisher: American Psychological Association.

[3] D. Badampudi, M. Unterkalmsteiner, and R. Britto, "Modern Code Reviews—Survey of Literature and Practice," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 4, 2023.

[4] A. S. Barroso, J. S. Madureira, M. S. Soares, and R. P. do Nascimento, "Influence of human personality in software engineering-a systematic literature review," in *International Conference on Enterprise Information Systems*, vol. 2. SCITEPRESS, 2017, pp. 53–62.

[5] R. F. Baumeister, E. Bratslavsky, M. Muraven, and D. M. Tice, "Ego depletion: Is the active self a limited resource?" *Journal of Personality and Social Psychology*, vol. 74, no. 5, pp. 1252–1265, 1998, place: US Publisher: American Psychological Association.

[6] R. Bee and F. Bee, *Constructive Feedback*. CIPD Publishing, 1998.

[7] G. Calikli and A. Bener, "Empirical analyses of the factors affecting confirmation bias and the effects of confirmation bias on software developer/tester performance," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, ser. PROMISE '10. New York, NY, USA: ACM, Sep. 2010, pp. 1–11.

[8] S. Chattopadhyay, N. Nelson, A. Au, N. Morales, C. Sanchez, R. Pandita, and A. Sarma, "A tale from the trenches: cognitive biases and software development," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. Seoul South Korea: ACM, Jun. 2020, pp. 654–665.

[9] J. A. O. G. da Cunha and H. P. de Moura, "Towards a substantive theory of project decisions in software development project-based organizations: A cross-case analysis of it organizations from brazil and portugal," in *2015 10th Iberian Conference on Information Systems and Technologies (CISTI)*. IEEE, 2015, pp. 1–6.

[10] S. Danziger, J. Levav, and L. Avnaim-Pesso, "Extraneous factors in judicial decisions," *Proceedings of the National Academy of Sciences*, vol. 108, no. 17, pp. 6889–6892, Apr. 2011.

[11] F. Ebert, F. Castor, N. Novielli, and A. Serebrenik, "An exploratory study on confusion in code reviews," *Empirical Software Engineering*, vol. 26, no. 1, p. 12, Jan. 2021.

[12] F. Fagerholm, M. Felderer, D. Fucci, M. Unterkalmsteiner, B. Marculescu, M. Martini, L. G. W. Tengberg, R. Feldt, B. Lehtelä, B. Nagyváradi, and J. Khattak, "Cognition in Software Engineering: A Taxonomy and Survey of a Half-Century of Research," *ACM Computing Surveys*, vol. 54, no. 11s, pp. 1–36, Jan. 2022.

[13] L. Faulkner, "Beyond the five-user assumption: Benefits of increased sample sizes in usability testing," *Behavior Research Methods, Instruments, & Computers*, vol. 35, no. 3, pp. 379–383, Aug. 2003.

[14] B. Fischhoff, *Debiasing*. Cambridge University Press, 1982, p. 422–444.

[15] M. Fleischmann, M. Amirpur, A. Benlian, and T. Hess, "Cognitive biases in information systems research: A scientometric analysis," *Tel Aviv*, p. 23, 2014.

[16] P. W. Gonçalves, E. Fregnan, T. Baum, K. Schneider, and A. Bacchelli, "Do explicit review strategies improve code review performance? Towards understanding the role of cognitive load," *Empirical Software Engineering*, vol. 27, no. 4, p. 99, Jul. 2022.

[17] S. D. Gunawardena, P. Devine, I. Beaumont, L. P. Garden, E. Murphy-Hill, and K. Blincoe, "Destructive Criticism in Software Code Review Impacts Inclusion," *Proceedings of the ACM on Human-Computer Interaction*, vol. 6, no. CSCW2, pp. 1–29, Nov. 2022.

[18] M. G. Haselton, D. Nettle, and D. R. Murray, "The Evolution of Cognitive Bias," in *The Handbook of Evolutionary Psychology*. John Wiley & Sons, Ltd, 2015, pp. 1–20.

[19] M. Jørgensen and E. Papatheocharous, "Believing is Seeing: Confirmation Bias Studies in Software Engineering," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, Aug. 2015, pp. 92–95.

[20] D. Kahneman, *Thinking, fast and slow*. Macmillan, 2011.

[21] D. Kahneman and A. Tversky, "Prospect theory: An analysis of decision under risk," *Econometrica*, vol. 47, no. 2, pp. 363–391, 1979.

[22] J. Klayman, "Varieties of Confirmation Bias," in *Psychology of Learning and Motivation*, J. Busemeyer, R. Hastie, and D. L. Medin, Eds. Academic Press, Jan. 1995, vol. 32, pp. 385–418.

[23] D. Laibson, "Golden Eggs and Hyperbolic Discounting*," *The Quarterly Journal of Economics*, vol. 112, no. 2, pp. 443–478, May 1997.

[24] B. Laugwitz, T. Held, and M. Schrepp, "Construction and Evaluation of a User Experience Questionnaire," in *HCI and Usability for Education and Work*, ser. LNCS, A. Holzinger, Ed., vol. 5298. Springer, 2008, pp. 63–76.

[25] L. M. Leventhal, B. E. Teasley, and D. S. Rohlman, "Analyses of factors related to positive test bias in software testing," *International Journal of Human-Computer Studies*, vol. 41, no. 5, pp. 717–749, Nov. 1994.

[26] J.-Y. Mao, K. Vredenburg, P. W. Smith, and T. Carey, "The state of user-centered design practice," *Communications of the ACM*, vol. 48, no. 3, pp. 105–109, 2005.

[27] R. Mohanani, I. Salman, B. Turhan, P. Rodriguez, and P. Ralph, "Cognitive Biases in Software Engineering: A Systematic Mapping Study," *IEEE Transactions on Software Engineering*, vol. 46, no. 12, pp. 1318–1339, Dec. 2020.

[28] D. Norman, *The design of everyday things: Revised and expanded edition*. Basic books, 2013.

[29] D. A. Norman and S. W. Draper, *User Centered System Design; New Perspectives on Human-Computer Interaction*. USA: L. Erlbaum Associates Inc., 1986.

[30] G. A. Pignatiello, R. J. Martin, and R. L. Hickman, "Decision fatigue: A conceptual analysis," *Journal of Health Psychology*, vol. 25, no. 1, pp. 123–135, Jan. 2020.

[31] A. Rainer and S. Beecham, "A follow-up empirical evaluation of evidence based software engineering by undergraduate students," *NA*, Jun. 2008.

[32] P. Ralph, "Introducing an empirical model of design," in *Proceedings of The 6th Mediterranean Conference on Information Systems*, 2011.

[33] C. Sadowski, E. Söderberg, L. Church, M. Sipko, and A. Bacchelli, "Modern code review: a case study at google," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, ser. ICSE-SEIP '18. ACM, May 2018, pp. 181–190.

[34] I. Salman, B. Turhan, and S. Vegas, "A controlled experiment on time pressure and confirmation bias in functional software testing," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1727–1761, Aug. 2019.

[35] D. M. Schrepp, "User experience questionnaire handbook - ueq-online.org," Dec. 2019.

[36] M. Schrepp, A. Hinderks, and J. Thomaschewski, "Applying the User Experience Questionnaire (UEQ) in Different Evaluation Scenarios," in *Design, User Experience, and Usability. Theories, Methods, and Tools for Designing the User Experience*, ser. LNCS, D. Hutchison, T. Kanade, J. Kittler, J. M. Kleinberg, A. Kobsa, F. Mattern, J. C. Mitchell, M. Naor, O. Nierstrasz, C. Pandu Rangan, B. Steffen, D. Terzopoulos, D. Tygar, G. Weikum, and A. Marcus, Eds., vol. 8517. Springer, 2014, pp. 383–392.

[37] ——, "Construction of a Benchmark for the User Experience Questionnaire (UEQ)," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 4, p. 40, 2017.

[38] ——, "Design and Evaluation of a Short Version of the User Experience Questionnaire (UEQ-S)," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 4, no. 6, p. 103, 2017.

[39] H. H. Sievertsen, F. Gino, and M. Piovesan, "Cognitive fatigue influences students' performance on standardized tests," *Proceedings of the National Academy of Sciences*, vol. 113, no. 10, pp. 2621–2624, Mar. 2016.

[40] D. Spadini, G. Çalikli, and A. Bacchelli, "Primers or reminders?: the effects of existing review comments on code review," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. Seoul South Korea: ACM, Jun. 2020, pp. 1171–1182.

[41] B. E. Teasley, L. M. Leventhal, C. R. Mynatt, and D. S. Rohlman, "Why software testing is sometimes ineffective: Two applied studies of positive test strategy," *Journal of Applied Psychology*, vol. 79, no. 1, pp. 142–155, 1994, place: US Publisher: American Psychological Association.

[42] A. Tversky and D. Kahneman, "Judgment under Uncertainty: Heuristics and Biases," *Science*, vol. 185, no. 4157, pp. 1124–1131, Sep. 1974.

[43] A. Waggoner Denton, "Improving the Quality of Constructive Peer Feedback," *College Teaching*, vol. 66, no. 1, pp. 22–23, Jan. 2018.