Do Explicit Review Strategies Improve Code Review Performance?

Pavlina Wurzel Gonçalves,1 Enrico Fregnan,1 Tobias Baum,2 Kurt Schneider,2 Alberto Bacchelli1

1University of Zurich, Switzerland 2Leibniz Universität Hannover, Germany 1<lastname>@ifi.uzh.ch 2<firstname>.<lastname>@inf.uni-hannover.de

ABSTRACT

Context: Code review is a fundamental, yet expensive part of software engineering. Therefore, research on understanding code review and its efficiency and performance is paramount.

Objective: We aim to test the effect of a guidance approach on review effectiveness and efficiency. This effect is expected to work by lowering the cognitive load of the task; thus, we analyze the mediation relationship as well.

Method: To investigate this effect, we employ an experimental design where professional developers have to perform three code reviews. We use three conditions: no guidance, a checklist, and a checklist-based review strategy. Furthermore, we measure the reviewers’ cognitive load.

Limitations: The main limitations of this study concern the specific cohort of participants, the mono-operation bias for the guidance conditions, and the generalizability to other changes and defects. Full registered report: https://doi.org/10.17605/OSF.IO/5FP7J; Materials: https://doi.org/10.6084/m9.figshare.11806656

ACM Reference Format:

1 INTRODUCTION

Code review is a widespread software engineering practice in which one or multiple reviewers inspect a code change written by a peer [1, 23] with the primary goal of improving software quality [5]. Performing a good code review is an expensive and time-consuming task [11]. Therefore, research is investigating how to improve code review efficiency and performance.

With this aim, researchers have developed many reading techniques to guide developers in reviewing code [15]. One of the guidance techniques commonly used in the industry is checklist-based reading [2, 8, 27]. A checklist guides a reviewer in what to look for.

Further guidance might be possible by telling a reviewer how to review. Providing a specific strategy to perform a development task has been proven to be helpful not only with scenario-based reading techniques for code inspection [2], but also in debugging or test-driven development [16, 20].

Checklists and strategies assist developers in performing complex tasks by systematizing their activity, thus lowering the cognitive load of reviewers [17, 20]. The result should be a more effective and efficient review.

In this experiment, we investigate the effect of guidance approaches on reviewing code. We compare three treatment groups – no guidance, checklist, and strategic checklist execution (strategy) - with the primary goal of improving software quality [3]. Performing a good review not only means finding many of the contained defects (effectiveness) but also finding them quickly (efficiency) [6]. We include not having any guidance as an additional control. Therefore, we ask:

RQ1: Does guidance in review lead to:
RQ1.1: a higher review effectiveness (share of functional defects found)?
RQ1.2: a higher review efficiency (functional defects found over the review time)?

We formalize our research question in the following hypotheses:

H1.1: there are significant differences in review effectiveness between checklist, strategy, and no guidance approach.
H0.1.1: there are no significant differences in review effectiveness between checklist, strategy, and no guidance approach.

H1.2: there are significant differences in review efficiency between checklist, strategy, and no guidance approach.
H0.1.2: there are no significant differences in review efficiency between checklist, strategy, and no guidance approach.

Both guidance approaches (checklist and strategy) systematize the activity of the reviewers by reducing the amount of information to keep in mind at a given time, thus, supposedly lowering developers’ cognitive load [20, 25]. Therefore, we investigate:

RQ2: Is the effect of guidance on code review mediated by a lower cognitive load?

We formalize our research question in the following hypotheses:
**3 EXPERIMENTAL STUDY**

The following section introduces the material we plan to use in this study; this material is publicly available [30].

**Experiment UI.** We employ a web-based tool that participants use to complete the experiment remotely. We log participants’ answers, environment, and UI interactions. The tool was built for our previous work and we modified it according to the new experiment’s requirements and past experience [5]. Figure 1 shows a partial view of the checklist implementation in the web-based experiment UI; a complete view is available in our online appendix [30].

**Checklist.** The checklist is developed based on recommendations in the literature and Microsoft checklists [24]. According to the literature, a good checklist requires a specific answer for each item, separates items by topic, and focuses on relevant issues [9, 12, 17]. Checklists should specify the scope in which items should be checked (e.g., “for each method/class”) to prevent developers from memorizing big portions of code and jumping through it [17]. Following these recommendations, we created a tentative version of the checklist.

For each seeded defect, the final checklist contains at least one item that helps to find the issue but does not give obvious clues about the type or location of the defects. To assess its face validity, we contacted three Java developers with experience in code review. Based on their feedback, we improved the items in our checklist. Then, we repeated this process with other three developers.

**Cognitive Load Questionnaire.** To measure cognitive load, we used a standardized questionnaire (StuMMBE-Q) [19]. It captures the two components of cognitive load (i.e., mental load and mental effort) in two 6-item subscales. Effort and difficulty ratings are reliable measures for the cognitive processing that contributes to cognitive load [13].

**System Usability Scale.** To measure the usability of the guidance approaches, we adapted the items of the System Usability Scale to fit the purpose of the checklist and strategy evaluation [7].

**3.3 Tasks**

We ask participants to perform three code reviews clarifying that they only have to look for functional defects in the code. The code changes to review are taken from a previous experiment on code review and contain both original and seeded defects [5]. All the changes are from the text editor jEdit, a system that was successfully employed in previous studies [5, 28]. The first short code change contains three defects, while the others contain nine and ten defects, respectively. The two large changes are presented to the participants in a randomized order. A description of the defects is publicly available [4].

Developers enter remarks in the code review UI by writing a comment in a text area that appears on any line the reviewer selects. A remark (Reported defect) is a note of the developer in natural language pointing at a place of a potential defect. As done previously [5], we count a remark as referring to a defect, if it is in the right position and can make a reader aware of the defect. One author rates all the remarks, a second author does the same for a subset and, then, the inter-rater agreement is computed.
Table 1: The variables of the study.

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Scale</th>
<th>Operationalization</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Independent variables (design):</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Guidance approach</td>
<td>The guidance provided by the tool to the participant: none, checklist, or strategy</td>
<td>nominal</td>
<td>see Sect. 3.2 and 3.3; randomized.</td>
</tr>
<tr>
<td><strong>Dependent variables:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of detected defects</td>
<td>Total detected functional defects, over all reviews by the participant (RQ1)</td>
<td>ratio</td>
<td>see Sec. 3.1</td>
</tr>
<tr>
<td>Review time</td>
<td>The needed net review time, i.e., the time needed for all the reviews subtracting pauses (RQ1)</td>
<td>ratio</td>
<td>Automatically measured by tool for each review task. Pauses are toggled by the participant.</td>
</tr>
<tr>
<td>Review effectiveness</td>
<td>Ratio of defects found by the participant over the total number of defects in the code change [6]</td>
<td>ratio</td>
<td>Computed at the end using the number of detected defects and the total number of defects.</td>
</tr>
<tr>
<td>Review efficiency</td>
<td>Number of defects found per hour spent reviewing [6]</td>
<td>ratio</td>
<td>Computed at the end using the number of detected defects and the review time.</td>
</tr>
<tr>
<td>Cognitive load</td>
<td>Load imposed on a person’s cognitive system while performing a particular task [25] (RQ 2)</td>
<td>ordinal</td>
<td>see Sect. 3.1 and 3.2</td>
</tr>
<tr>
<td><strong>Treated/Measured variables:</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prof. development experience</td>
<td>Years of experience with professional software development</td>
<td>ordinal</td>
<td>Measured: 6-point scale (&quot;never&quot; … &quot;11 years or more&quot;), questionnaire</td>
</tr>
<tr>
<td>Java experience</td>
<td>Years of experience with the Java programming language</td>
<td>ordinal</td>
<td>Measured: 6-point scale (&quot;never&quot; … &quot;11 years or more&quot;), questionnaire</td>
</tr>
<tr>
<td>Code review experience</td>
<td>The number of years of experience with code reviews</td>
<td>ordinal</td>
<td>Measured: 6-point scale (&quot;never&quot; … &quot;11 years or more&quot;), questionnaire</td>
</tr>
<tr>
<td>Current program. practice</td>
<td>How often the participant currently programs</td>
<td>ordinal</td>
<td>Measured: 5-point scale (&quot;not&quot; … &quot;daily or more often&quot;), questionnaire</td>
</tr>
<tr>
<td>Current code review practice</td>
<td>How often the participant currently performs code reviews</td>
<td>ordinal</td>
<td>Measured: 5-point scale (&quot;not&quot; … &quot;daily or more often&quot;), questionnaire</td>
</tr>
<tr>
<td>Fitness</td>
<td>Perceived tiredness or fitness of the participant</td>
<td>ordinal</td>
<td>Measured: 5-point scale (&quot;very tired&quot; … &quot;very fit&quot;), questionnaire</td>
</tr>
<tr>
<td>Experience with jEdit</td>
<td>Whether the participant has experience with the jEdit editor, the source of the changes to review</td>
<td>ordinal</td>
<td>Measured: 3-point scale (&quot;none&quot;, &quot;used&quot;, &quot;contributed&quot;), questionnaire</td>
</tr>
<tr>
<td>Code change</td>
<td>Code change under review, including the contained defects</td>
<td>nominal</td>
<td>Design: Each reviewer has to review three different code changes</td>
</tr>
<tr>
<td>Change part order</td>
<td>Order of presenting the code changes</td>
<td>nominal</td>
<td>For the two larger changes, the order is randomly chosen from 4 possibilities (see [5])</td>
</tr>
<tr>
<td>Usability</td>
<td>Perceived efficiency, effectiveness and satisfaction in the use of an object [7].</td>
<td>ordinal</td>
<td>Measured: 5-point scale (&quot;agree&quot; … &quot;disagree&quot;), questionnaire</td>
</tr>
<tr>
<td>Understandability</td>
<td>Participants’ understanding of the code. Number of correct answers over all answers per change</td>
<td>ratio</td>
<td>Measured: Multiple choice questions on the reviewed code change (available in [30])</td>
</tr>
</tbody>
</table>

3.4 Participants

The sample consists of about 100 developers from an Indian software development company normally hired for outsourcing projects. This company provides a wide range of services (e.g., web development, mobile development, and DevOps) and has more than 2,000 employees. We contacted the company via email and agreed to conduct the experiment in multiple iterations. Proceeding this way, we can adjust the experiment setup, if necessary: e.g., asking for participants with longer programming experience.

The experiment participants are going to be randomly assigned to one of the three conditions (no guidance, checklist, and strategy). The review tasks are performed in fulfillment of an official contract with the company and, therefore, we expect developers to complete them as part of their official working duties. In case we detect irregularities or drop-outs, we will ask the company to provide replacements.

We performed a power analysis to estimate the sample size needed to identify existing differences between the treatment groups. Based on previous studies, we do not expect a large effect size to appear [14]. The sample size is calculated using a standard setting for an ANOVA medium effect size [10]. The estimated total sample size is 66 participants.

The selection of the sample of developers who take part in the experiment is up to the project manager of the company. However, we are going to specify the number of participants in each iteration of the experiment and the required level of experience with Java.

Pilot. We initially contacted the aforementioned company with the aim of expanding our previous experiment on the effects of ordering code changes on review effectiveness and efficiency [5].
We conducted a pilot with 29 developers who did not receive any guidance in their review. Unexpectedly, the participants were not able to identify many defects. This finding triggered the idea to investigate guidance approaches to help reviewers become more effective. Moreover, it allowed us to test the experiment platform as well as the collaboration setup with the company.

3.5 Execution Plan

Participants access the experiment online via a provided URL. The assignment is rolled out in batches of 10 to 20 developers, to check for problems after each batch. The tool presents the following workflow to each participant:

Welcome. A welcome page, containing general information about the experiment and indications on how it takes place. We ask for informed consent and explicitly request the developers not to share information about the experiment with each other. We explain that participation is voluntary and can be canceled at any time.

Explanation. An explanation of the review tasks. It emphasizes that participants should focus on finding functional issues. We also give a brief explanation of the relevant support technique and an introduction to the software system to review.

Three code review tasks. In addition to the review tasks, the participant is presented with a questionnaire to measure cognitive load at the end of each task.

Usability. If the participant is assigned to the checklist or strategy group, they are presented with a questionnaire to measure cognitive load at the end of each task.

Demographics. General demographics questions: to gain a deeper understanding of the participants in each treatment group, we ask them general questions about potential confounders.

Closing. Final optional remarks on the experiment and ending.

3.6 Analysis Plan

Data Cleaning. We remove participants who do not finish all the reviews. We regard participants who spent less than 5 minutes on a review and entered no review remark as ‘did not finish’. We also exclude participants who restart the experiment or participate several times (we collect client IPs—hashed to guarantee data anonymization—and cookies). Furthermore, we control the developers’ comprehension of the system by asking questions about the change as code comprehension is an important condition for good reviews [1, 26]. We do not plan to remove other outliers unless we find specific reasons to believe the data is not valid.

Descriptive Statistics. We present descriptive statistics for the demographic, dependent, and independent variables per each treatment group by reporting means and standard deviations of respective variables. We present a correlation matrix table to assess potential covariance and relationships between examined variables.

Inferential Statistics. We take a frequentist stance. To verify our hypotheses for RQ1, we will conduct a One-way ANOVA to identify whether a difference between the three treatment groups exists and report the effect size. We will explore these differences further by using Tukey’s Range Test to do the post-hoc analysis. We use mediation analysis [29] for RQ2. We assess whether there is a partial or full mediation [21, 22] of the examined relationship by cognitive load using the mediation R package.

Validity Threats. The implementation of the treatments might be a validity threat. To check it, we measure the checklist/strategy usability. Furthermore, the validity of the study might be undermined by our choice of changes to review. To control for overly complicated changes, we ask developers about their comprehension of the code. All the participants work in the same company and have, at the best of our knowledge, a very similar technical and cultural background; this limits the generalizability of our findings.

4 ACKNOWLEDGMENTS

A. Bacchelli, P. Wurzel Gonçalves, and E. Fregnan gratefully acknowledge the support of the Swiss National Science Foundation through the SNF Projects No. P00P2_170529. P. Wurzel Gonçalves also acknowledges the student sponsoring support by CHOOSE, the Swiss Group for Software Engineering.
REFERENCES


