Assessing the Impact of File Ordering Strategies on Code Review Process

Farid Bagirov
farid.bagirov@jetbrains.com
JetBrains Research
Paphos, Cyprus

Pouria Derakhshanfar
pouria.derakhshanfar@jetbrains.com
JetBrains Research
Amsterdam, The Netherlands

Alexey Kalina
alexey.kalina@jetbrains.com
JetBrains
Munich, Germany

Elena Kartysheva
elena.kartysheva@jetbrains.com
JetBrains
Paphos, Cyprus

Vladimir Kovalenko
vladimir.kovalenko@jetbrains.com
JetBrains Research
Amsterdam, The Netherlands

ABSTRACT

Popular modern code review tools (*e.g.,* Gerrit and GitHub) sort files in a code review in alphabetical order. A prior study (on open-source projects) shows that the changed files' positions in the code review affect the review process. Their results show that files placed lower in the order have less chance of receiving reviewing efforts than the other files. Hence, there is a higher chance of missing defects in these files. This paper explores the impact of file order in the code review of the well-known industrial project IntelliJ IDEA. First, we verify the results of the prior study on a big proprietary software project. Then, we explore an alternative to the default *Alphabetical* order: ordering changed files according to their code diff. Our results confirm the observations of the previous study. We discover that reviewers leave more comments on the files shown higher in the code review. Moreover, these results show that, even with the data skewed toward *Alphabetical* order, ordering changed files according to their code diff performs better than standard *Alphabetical* order regarding placing problematic files, which needs more reviewing effort, in the code review. These results confirm that exploring various ordering strategies for code review needs more exploration.

CCS CONCEPTS

• **Software and its engineering → Empirical software validation**.

KEYWORDS

Code Review, Cognitive Bias, Ranking Metrics

ACM Reference Format:
Farid Bagirov, Pouria Derakhshanfar, Alexey Kalina, Elena Kartysheva, and Vladimir Kovalenko. 2023. Assessing the Impact of File Ordering Strategies on Code Review Process. In *Proceedings of International Conference on Evaluation and Assessment in Software Engineering (EASE 2023).* ACM, New York, NY, USA, 4 pages. https://doi.org/XXXXXXX.XXXXXXX

EASE 2023, June 14–16, 2023, Oulu, Finland

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Code review is one of the common software engineering practices. Its purposes are to identify defects in newly implemented code and improve quality of the software under development [2–4]. A recent study by Fregnan *et al.* [6] shows that the order of changed files in a code review impacts the quality of the review process and, thereby, indirectly affects the quality of the code. They also show that the files ranked higher in the order get more comments compared to the lower ones. Moreover, they conducted an experiment with handcrafted examples confirming that bugs in lower files are less likely to be noticed. Given these findings, analyzing the ordering strategies and identifying the most beneficial way of arranging changed files in the code review is helpful for developers.

In this study, we assess different file ordering strategies on 51,566 recent code reviews mined from the IntelliJ IDEA project [1]. At the first step, we verify results from [6] on this big industrial closed-source project. We analyze the correlation between the location of each changed file in a code review and the number of comments it received. We also compare the *Alphabetical* order (which is used by most of the popular code review frameworks *e.g.,* GitHub, Gerrit, and JetBrains Space) against the *Random* order (as the baseline) to confirm the findings of the previous study [6].

Additionally, we perform a detailed analysis to compare the *Alphabetical* order against *Code Diff* (as an alternative file ordering strategy). In this experiment, we first compare these two strategies on all code reviews. Then, we take a deeper look and explore the capability of these two strategies on ordering different types of files: (i) Java files, (ii) Kotlin files, and (iii) files that are commented only at the later iterations of a review.

We compare these strategies in terms of their capability to position *problematic* files (*i.e.,* files that require more attention and activity from the reviewers) at the top. Alike to the previous study [6], we consider files that are commented during the review process as the problematic files. We use three common ranking metrics to measure the accuracy of file ordering strategies in putting the problematic files higher: *MRR*, *prec@k*, and *nDCG*.

Similarly to the prior study by Fregnan *et al.* [6], our results show a significant weak correlation between the position of files and the number of comments they receive. The results also show

that, according to all the ranking metrics, *Alphabetical* order significantly outperforms the *Random* ordering with small effect sizes. Moreover, according to two ranking metrics (*MRR* and *nDCG*), *Code Diff* is significantly better than *Alphabetical* ordering strategy in organizing all of the files in the code review. Interestingly, this outperformance is more evident when ordering Kotlin files. In this case, *Code Diff* significantly outperforms *Alphabetical* ordering, according to all ranking metrics, with a medium effect size. However, we do not see this considerable difference in ordering Java files and files commented at the later iterations.

The remainder of this paper is structured as follows: Section 2 explains the methodology of our experiment. Section 3 presents our results and findings. Finally, Section 4 describes the conclusion and our future works.

2 METHODOLOGY

This section presents the design of our experiments, the ordering strategies we compare, the code review selection, the metrics we use to compare the ordering strategies, and our data analysis procedure.

2.1 Experiment Design

In our study, we seek to answer the two primary questions:

Question 1. Does the order of files impact the number of comments each file receives in the IntelliJ IDEA project? For this, as was done in [6], we calculate Spearman's correlation test (a non-parametric correlation test that shows how well the relationship between two random variables be described as a monotonic function [9]) on the number of comments and position of the files. Moreover, we compare the default *Alphabetical* order and the *Random* order. Since the *Alphabetical* order is arbitrary with regards to problematic files, comparison with random order can show whether there is a bias in what files get comments. To compare both orders, we calculate ranking metrics *MRR*, *prec@k*, and *nDCG* (see Section 2.4) to assess the capability of these strategies in ordering commented files. Then, we compare these two ordering strategies via Student's t-test (see Section 2.5).

Question 2. Can an order alternative to *Alphabetical* improve code review process? For this, we analyze the *Code Diff* order, which sorts files according to the number of changes in them. Similarly to the previous setting, we compare *Code Diff* against *Alphabetical* with ranking metrics and Student's t-test (see Sections 2.4 and 2.5). We also look separately at how orders place problematic Kotlin files, problematic Java files, and problematic files missed during the first review iterations.

2.2 File orders

Alphabetical. This is the default order in most code review systems, including one used in JetBrains. This ordering strategy positions files based on their full path in alphabetically ascending order.

Random. We use this order as a baseline. Since the *Alphabetical* order is rather arbitrary, the comparison with the true random order can shed some light on the bias in the number of comments. In the *Random* order, files are shuffled randomly.

Code Diff. This ordering strategy organizes files according to the number of lines added or removed in them in descending order (*i.e.,* files in which more lines are added and removed have a higher priority in this strategy).

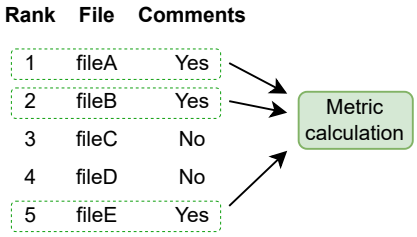

Figure 1: Metric calculation for a single code review.

2.3 Code Review Selection

Our target project was IntelliJ IDEA [1] — a large proprietary software product developed by JetBrains. For this project, we mined all code reviews from 01.01.2020 to 28.02.2023. This resulted in 51,566 different code reviews. Then, we excluded all code reviews that are insensitive to the file ordering strategy (*i.e.,* reviews without comments to the modified files and reviews with only a single changed file). For each remaining review, we collect the list of modified files, files that received at least one comment (they are treated as problematic files in our evaluation), and the number of comments they received. We use this data to calculate the ranking metrics using equations presented in Section 2.4. Figure 1 presents an overview of metric calculation using the collected data.

For the *Code Diff* order, we needed to collect file changes. However, as code review is an iterative process, the reviewers' comments often cause some changes in files within the code review. To correct for this, for each code review, we only consider code changes made before the first comment. After that, we remove the code reviews without comments on the changed files or with only one changed file. We conducted our experiments on the 9475 resulting reviews. Their statisctis are presented in 'Whole data' column in Table 1.

To compare the *Code Diff* and *Alphabetical* order (Question 2), we additionally look at a specific subset of the problematic files. We consider three subsets: Kotlin files, Java files, and files missed in the first review iterations. To identify Kotlin and Java files, we look at the file extension (`.java` for Java, and `.kt` or `.kts` for Kotlin). We consider files that were missed in the first review (called *missed files* hereafter) as files that changed early but were commented only on the later iterations of the code review. To identify them, we collect the time each file is commented and the latest time it is changed during the review process before the first comment. Between these two moments, if we see any reviewing and committing activities on any changed file, we consider the file as *initially missed.* Then, we filter out all code reviews without problematic files. We compare ordering strategies on the remaining reviews according to the positions of the remaining problematic files (*e.g.,* count all metrics only on Kotlin files with comments). Table 1 presents the data statistics for Kotlin, Java, and missed files subsamples in the respective columns.

Table 1: Statistics of the final data used in our study. The 'Whole data' column presents statistics for the whole filtered data. The 'Missed files' column contains the statistics of problematic files that are missed in the first iteration of code reviews. The 'Java' and 'Kotlin' columns present the statistics of problematic Java and Kotlin files, respectively.

	Whole data	Missed files	Java	Kotlin
Number of code reviews	9475	235	5145	4075
Average number of commented files	1.71	1.54	1.47	1.58
Average number of modified files	14.82	21.20	16.26	13.91
Average number of added lines per code review	572.36	711.54	723.70	389.88

2.4 Metrics

We use common ranking metrics to compare different orders. With our metric selection, we tried to cover different aspects of the ordering strategies. We choose *MRR* to represent the positions of the first problematic files, *prec@k* to show the number of problematic files appeared at the top, and *nDCG* to show full ranking's quality.

MRR. Reciprocal rank is the inverse position of the first commented file. Mean reciprocal rank, or *MRR*, is the average of reciprocal ranks on all code reviews.

Precision@k (prec@k). For a single code review, precision@k is a portion of commented files in the top k. For the whole dataset, precision@k is the average precision on each code review. We choose a commonly used value of $k = 10$ to represent how order strategies work for the several files on the top.

nDCG. Discounted cumulative gain (DCG) measures the cumulative 'usefulness' of the problematic files based on their position [8]. If there is a commented file at position i, it provides a gain of $\frac{1}{\log i+1}$. Normalized discounted cumulative gain (nDCG) is DCG that is normalized to the maximum value that DCG can be.

Metric calculation. For *Alphabetical* and *Code Diff* orders, we find the positions of commented files and calculate metrics based on them. For the *Random* order, we can analytically calculate the expected values of the metrics without sampling actual positions.

Below, we present calculations for metric values done on a single code review. n denotes number of modified files, m number of files with comments, and CF is the set of commented files.

- **MRR**. Let denote hr as the rank of the highest commented file in the order. Then, we calculate expectation of MRR in general form: $\sum_{hr=i} MRR(hr = i)p(hr = i) = \sum \frac{p(\text{hr=i})}{i}$. We estimate $p(hr = i) = p(hr \le i) - p(hr \le i)$. We calculate the latter probability with the help of hypergeometric distribution HG with parameters n, m, i as $p(hr \le i) = 1 - HG(x = 0)$. Hypergeometric distribution with parameters n, m, i is a discrete probability distribution that defines the probability of drawing x objects of a specific type in i draws from n object, where, in total, there are m objects with the desired type. In our case, we count the complementary probability of not having any commented files in the first i.
- **Prec@k** is calculated with the following equation:

$$Prec@k = \frac{1}{\min(k,n)} \sum_{f \in CF} \mathbf{E}[\text{f in top k}] = \frac{1}{\min(k,n)} \sum_{f \in CF} \frac{\min(k,n)}{n} = \frac{m}{n}$$

- Normalisation in **nDCG** is not affected by randomness, and only DCG should be calculated by the following equation:

$$DCG = \sum_{f \in CF} \mathbf{E} \frac{1}{\text{position of } f + 1} = \frac{m}{n} \sum_{i=1}^{n} \frac{1}{i+1}$$

2.5 Data Analysis

*Metric comparison (To answer both **Question 1** and **Question 2**).* All the metrics we use in this study are averaged over all code reviews. Therefore, we can consider them as an expected value over some code review distribution. For comparison, we calculate each ranking metric for each code review and use paired Student's t-test, with $\alpha = 0.05$ for Type I errors, to assess the significance of the results ($p - value \le 0.05$ indicates the significance of our results). We use Student's t-test for two related samples since we need to compare the mean values of two samples. Moreover, since the number of samples is high due to CLT mean, values of the metrics approximately have a normal distribution [7]. We use Cohen's d test, which is standard for t-test [5], to calculate effect sizes (we treat effect sizes < 0.2 as small and from 0.2 to 0.5 as medium).

*Correlation test (To answer **Question 1**).* To confirm the correlation between the file's position and the number of comments it gets, we perform the Spearman correlation test [9]. We perform this correlation test on a normalized number of comments and file positions. We normalize the number of comments to a file to the total number of comments in the code review, and the file position is normalized to the total number of files in the review. With normalization, we remove the effect of reviews of different sizes (*i.e.,* positions with high numbers have fewer comments since they can appear only in a small portion of code reviews).

3 FINDINGS AND RESULTS

3.1 Question 1.

Correlation between position and number of comments. We obtained p-value < 0.001 and $\rho = -0.121$, which aligns with the results of Fregnan *et al.* [6]: there is a correlation between the number of comments and position, but a rather small one.

Table 2: Comparison of *Alphabetical* order and *Random* order on the whole data. Bold highlights values that are significantly higher than their counterpart (p-value < 0.05).

	MRR	prec@k	nDCG
***Alphabetical* order**	**0.537**	**0.285**	**0.646**
***Random* order**	0.491	0.278	0.607
effect size	0.158	0.145	0.187

Table 3: Comparison of *Alphabetical* order and *Code Diff* order on the whole data. Bold highlights values significantly higher than their counterpart (p-value < 0.05).

	MRR	prec@k	nDCG
***Alphabetical* order**	0.537	**0.285**	0.646
***Code Diff* order**	**0.560**	0.283	**0.657**
effect size	0.06	0.04	0.04

Table 4: Comparison of *Alphabetical* order and *Code Diff* order on problematic file that were missed in the first iteration of code reviews.

	MRR	prec@k	nDCG
***Alphabetical* order**	0.364	0.175	0.505
***Code Diff* order**	0.370	0.174	0.516
effect size	0.01	0.02	0.04

Alphabetical order vs Random order We present the results of this comparison in Table 2. In the rows "*Alphabetical* order" and "*Random* order", we list metric values for the *Alphabetical* and *Random* orders, respectively. Metric value in bold highlights the significantly better order strategy (*i.e.,* p-value < 0.05). This table also reports p-values and effect sizes of point-wise metric comparisons in the respective rows. The results show that the *Alphabetical* ordering places the commented files significantly higher, but the effect size is small (< 0.2). Since both orders are arbitrary concerning actual problematic files, we can conclude that there is a small bias towards the *Alphabetical* order. This result aligns with the results from the correlation test and also indicates that the files that are higher tend to get more comments.

3.2 Question 2.

Alphabetical order vs Code Diff order. Table 3 shows the comnparison between these two ordering strategies. Despite the bias toward the *Alphabetical* order, *Code Diff* significantly outperforms *Alphabetical* in *MRR* and *nDCG*. This can be interpreted as *Code Diff* order being better in showing the first problematic file higher (*MRR*) and has a better ranking as a whole (*nDCG*). However, despite the difference, the values of effect sizes are low (< 0.2). This can be caused by heavy bias from the *Alphabetical* order or by low sensitivity of the selected metrics. Verification of these hypotheses is our plan for the future work.

Additionally, we compared the *Alphabetical* order and the *Code Diff* order of different types of *problematic* files (Tables 4, 6, and 5). Mostly, there is no significant difference between *Alphabetical* and *Code Diff* order. However, the *Code Diff* order positions Kotlin files higher with the relatively larger effect size (> 0.2). It is worth noting, that results reported here can underestimate real effect, since historical data were sorted in *Alphabetical* order.

4 CONCLUSION AND FUTURE WORK

The order of files in code review impacts the level of attention they receive from reviewers. Hence, using a file ordering strategy that places the files likely to require more reviewing effort on top of the list is potentially beneficial.

Table 5: Comparison of *Alphabetical* order and *Code Diff* order on Kotlin problematic files.

	MRR	prec@k	nDCG
***Alphabetical* order**	0.448	0.251	0.583
***Code Diff* order**	**0.538**	**0.253**	**0.639**
effect size	0.23	0.03	0.21

Table 6: Comparison of *Alphabetical* order and *Code Diff* order on Java problematic files.

	MRR	prec@k	nDCG
***Alphabetical* order**	0.534	**0.264**	0.644
***Code Diff* order**	0.544	0.261	0.645
effect size	0.02	0.05	0.01

In this work, we have applied three different file ordering strategies (*Alphabetical*, *Random*, and *Code Diff* ordering) on 51,566 code reviews mined from the IntelliJ project. We have evaluated these ordering techniques in terms of their capability of placing problematic files at the top of the order. For this evaluation, we have used three ranking metrics: *MRR*, *nDCG*, and *prec@k*. Our results confirm the correlation between the positions of changed files in the code review and the number of comments they receive. We also show that an alternative ordering technique (*Code Diff* ordering) outperforms the standard *Alphabetical* ordering.

Our results suggest that evaluating and identifying a file ordering strategy better than standard *Alphabetical* order requires more exploration. Hence, in our future work, we aim to compare ordering in the unbiased setting and explore other promising ordering strategies (*e.g.,* using large language models to identify problematic files). Additionally, we plan to test promising ordering strategies in actual workflows and evaluate their effect on the software quality.

REFERENCES

[1] [n. d.]. *IntelliJ IDEA*. [Online; accessed 10-March-2023].
[2] A. Frank Ackerman, Lynne S. Buchwald, and Frank H. Lewski. 1989. Software inspections: an effective verification process. *IEEE software* 6, 3 (1989), 31–36.
[3] Tobias Baum and Kurt Schneider. 2016. On the need for a new generation of code review tools. In *Product-Focused Software Process Improvement: 17th International Conference, PROFES 2016, Trondheim, Norway, November 22-24, 2016, Proceedings 17*. Springer, 301–308.
[4] Tobias Baum, Kurt Schneider, and Alberto Bacchelli. 2017. On the optimal order of reading source code changes for review. In *2017 IEEE international conference on software maintenance and evolution (ICSME)*. IEEE, 329–340.
[5] Jacob Cohen. 2013. *Statistical power analysis for the behavioral sciences*. Routledge.
[6] Enrico Fregnan, Larissa Braz, Marco D'Ambros, Gül Çalıklı, and Alberto Bacchelli. 2022. First come first served: the impact of file position on code review. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 483–494.
[7] Arthur Stanley Goldberger et al. 1964. *Econometric theory*. (1964).
[8] Kalervo Järvelin and Jaana Kekäläinen. 2002. . *ACM Transactions on Information Systems (TOIS)* 20, 4 (2002), 422–446.
[9] Jerome L Myers, Arnold D Well, and Robert F Lorch. 2013. *Research design and statistical analysis*. Routledge.