Does Peer Code Review Change My Mind on My Submission?

Sven Strickroth
sven.strickroth@ifi.lmu.de
LMU Munich
München, Germany

ABSTRACT

Peer review can be used as a collaborative learning activity in which people with similar competencies evaluate other students' submissions and/or provide feedback. It provides many potential benefits such as timely feedback, high motivation, reduced workload for teachers, collaboration among the students, improving the code, and seeing other solution strategies. However, there are also challenges and contradictory results such as low motivation, participation, quality, and no improvements in the reviews. This article attempts to shed more light on these issues through an empirical investigation in a university-based introductory programming course with approx. 900 students. In the evaluation, this paper empirically investigates the effects of reviewing other solutions on the view of one's own solution and how students can be motivated to regularly work on voluntary homework assignments. Furthermore, there is an analysis of the peer reviews regarding their quality (length and correctness), and the students' participation and perceptions. The results indicate that giving feedback can change the view on one's own submission regarding the complete correctness, the majority of feedback is rather short, peer review assignments are a major driver for working on the assignments, and the majority of students like seeing other solutions. The majority of students seems to be able to identify correct submissions as correct, however, (partly) incorrect submissions are also often classified as completely correct. Possible measures to address these weaknesses are discussed.

CCS CONCEPTS

• **Social and professional topics → Computing education**; • **Applied computing** → *Collaborative learning*.

KEYWORDS

peer review, peer code review, code review, peer teaching, feedback, programming education, mass education

ACM Reference Format:
Sven Strickroth. 2023. Does Peer Code Review Change My Mind on My Submission?. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3587102.3588802

1 INTRODUCTION

Learning programming is often perceived as hard by novices [22]. Working on exercises and timely feedback are important for learning [12]. Furthermore, learning is, according to the constructivism learning theory, often seen as an inherently social process. In continental Europe, however, higher education is often dominated by mass education and large classes attended by several hundred students. A general problem in such courses is that social interactions, discussions, and feedback become rather limited. First, providing all students manual feedback for their submissions in a timely manner is not possible due to the teacher to student ratio [28, 33]. Second, a missing or reduced exchange with others (fellow students, teaching assistants, or professors) was an intensified issue during the Corona pandemic and its lockdowns.

In mass education students are often on their own and need self-regulated learning skills to motivate themselves to work on the homework assignments and to learn for examinations [13]. Therefore, students often do not work on homework assignments (for various reasons). To address these issues, homework assignments are frequently declared a condition of participation in an examination or the overall grading is based on both homework and an examination. However, these approaches might not be possible due to examination regulations or might cause excessive discussions such as when students feel mis-graded.

One possibility to address these issues is to use *peer review* or *peer feedback*. This technique does not only help to cope with an insufficient number of teachers but also makes students collaborate as well as to read and critique code. Students can see different solution strategies and are actively and directly involved to evaluate and deliver written feedback on the work of their peer students [23] – at best the reviewers as well as the reviewees profit.

This paper describes an approach to conduct peer review in large classes, provides an analysis of the approach and the peer reviews, and it answers the following research questions:

- RQ1: How motivating are peer reviews to work on (voluntary) homework assignments?
- RQ2: How can the delivered peer reviews be characterized (regarding length, correctness, etc.)?
- RQ3: Does seeing other submissions change the view on one's own submission?
- RQ4: How do students perceive the peer review?

The remainder of this paper is organized as follows: First, the related research on automatic feedback and peer (code) reviews is outlined. Second, the evaluation setting is described. Third, the dataset, evaluation method, and results are presented. The paper closes with a discussion, a summary, and an outlook.

2 RELATED RESEARCH

To support teaching in large classes, many (semi-)automated e-assessment systems have been developed [32, 19]. An advantage of such tools is that timely feedback can be provided 24/7. Disadvantages are, however, that all tests and feedback need to be pre-defined, there seems to be a focus on functional correctness [19, 7, 21], and not all aspects such as creativity or implementation quality can be assessed automatically [20]. Semi-automatic e-assessment systems can pre-correct and significantly speed up the correction [31, 30] but providing good feedback still requires a significant amount of time and do not alleviate discussions on possible mis-grading. Hence, automated systems cannot be the single solution.

In education, peer review is a collaborative learning activity in which people with similar competencies evaluate fellow students' submissions and/or provide feedback. Peer review has shown to be a powerful method as it allows students to receive more timely and extensive feedback, and to exchange ideas [10, 11, 15, 28]. Multiple reviews by different peers can be better than a single review of an expert [6]. Also, a positive impact on learning regarding a longer time learners spent on a subject and reflection on one's own learning triggered by the review have been reported [34, 24].

Using peer review for programming education dates back at least two decades [38, 15]. A recent systematic literature by Indriasari et al. summarizes benefits and barriers for effective peer code review [15]: Students are often motivated to do peer reviews [27, 9, 26]. Peer review was often perceived as a valuable experience by the students where they can e. g. exchange ideas, see alternative solution strategies, learn to give and receive criticism, learn from feedback, and collaborate [15]. Also, the quality of the peer reviews and their ability to detect errors can improve over time [29, 5] as well as the quality of the code: There are reports that peer reviews can correct low-level syntax errors, improve quality, and promote discussions of higher-level design and implementation issues [14, 18]. Furthermore, students reassess their own solution attempt [37]. Despite all the benefits, a low level of enthusiasm/motivation/engagement is also often reported as a primary problem [15, 13, 36, 29]. The participation rate in the peer review can be very low (e. g., 15 % [13]). There is research on using gamification to increase motivation [16]. Also, the length of the feedback (e. g., 14 words on average [13]) and the quality of the reviews can be very low (e. g., 28 % of the feedback being partly incorrect [13]). Reported reasons are an actual or perceived lack of student understanding [36] or that students just rush through the code at the last minute [29]. There are also reports of no improvements in review skills over time [35].

In summary, the use of peer review has great pedagogical benefits. However, for quality and review skills, there are contradictory results reported. According to Indriasari et al. [15], the majority of evaluations rely on evidence based on self-reported student opinions. They state that experimental evidence is largely missing to validate results and more research is needed regarding the correctness of the peer reviews, and reward and penalty schemes [15]. Also, most peer review settings known by the author comprise less than 200 students. Therefore, this paper addresses the research gap to empirically investigate whether seeing other solutions changes the view on one's own solution, analyzes the peer reviews, and evaluates an approach to motivate students to participate regularly.

3 EVALUATION SETTING

The setting for this research was the first semester computer science course “Introduction to Programming” (programming language is Java) at Ludwig Maximilian University of Munich, Germany, of winter term 2021/2022. This course was attended by approx. 900 students with computer science as a major (or minor). There were two 90-minute lectures a week, and 90-minute exercise sessions conducted by 18 student teaching assistants (TA) in which the homework assignments were discussed in the following week after the deadline. The student TAs were not involved in the peer reviews. During the semester there were 10 (weekly with three exceptions $>$ 1 week) assignment sheets containing 3 to 5 tasks. In general, there are argumentation, programming, modelling with UML, and multiple-choice/cloze tasks. The first two sheets contained only argumentation and multiple-choice/cloze tasks. All following sheets contained at least two programming tasks (median $m = 3$). Working on the assignments and peer reviews was fully voluntary and ungraded. Solution attempts could be submitted to the open-source GATE system [31] where a syntax and a (blackbox) function test were available. Each test could be requested twice to prevent abusing the system such as gaming the system approaches (cf. [2]). Until the deadline, unlimited (re-)submissions were possible.

Based on the recommendations by Indriasari et al. [15], there is only one peer review task per exercise sheet and there is at least a whole week allocated for delivering the review giving the students enough time for proper reviews. After the deadline of an exercise sheet, the peer reviews were assigned randomly. For the peer review tasks there was no automatic feedback available in order to be able to investigate the students' evaluation skills. During the review, the students had to assess two fellow students' submissions on a 4-point Likert scale (4=best) according to the correctness, completeness, readability (referring to style), and comprehensibility of the approach to the solution, and had to write a free comment (the students were explicitly asked to give (positive) feedback and hints for possible improvements). The review process was quite strict: First, the students had to rate their own submission, then they needed to review two other submissions, and finally, had to rate their own submission again. The reviews became automatically visible to students after the deadline. The first two sheets were mainly intended for getting familiar with the peer review process (no extra training); all following reviews were on tasks with coding.

Peer review requires active participation and the no-feedback rate should be as low as possible. Hence, students need to participate regularly. Not delivering a submission or review twice resulted in an exclusion from the peer review for the rest of the term. Viewing received reviews was possible even without delivering. This strict exclusion rule was softened after the Christmas break (starting with sheet 8) and students could freely decide for each assignment whether they want to participate.

4 EVALUATION AND RESULTS

To investigate the research questions, GATE stores the peer reviews, the self-assessments, the submission, exam scores, and meta-data on the system usage (e. g., time of delivery and time needed for the reviews and self-assessments). There is no randomized control-group design for ethical reasons. For the time measurement a timestamp was included in the HTML form and then subtracted from the current timestamp when processing the HTTP-POST request. This is not an optimal way regarding outliers, but should allow a rough evaluation. Furthermore, syntax and unit tests were run on the the submissions to compare the test results with the peer review ratings. All these data are used for the following quantitative analyses.

All students were explicitly asked whether they voluntarily agree that their anonymized submissions, peer reviews, and exam grades can be used for research. The dataset consists of 695 students who agreed and delivered 3050 submissions (i. e., 69 % of the unfiltered dataset) and had 4680 peer reviews assignments (58 %).

At the end of the semester there was a questionnaire comprising closed and open-ended questions regarding the peer-review and the usage of the GATE system. For the four open-ended questions presented in this paper a Thematic Analysis [4, 25] was performed. All comments were tagged, and the frequencies were counted.

4.1 Participation

Tab. 1 gives an overview of the peer review assignments and the associated number of submissions, review tasks, completed reviews, students eligible to participate in the reviews, and some characterizations that will be discussed in the following sections.

For each exercise sheet the peer reviewed assignment is the one with the most submissions (statistically significant difference for each sheet and in total: UTest $p < .01$, non-parametric test, cf. [3, 17]; avg. factor 1.53 more submissions compared to other programming assignments) – followed by multiple choice assignments (avg. factor 1.2). Tab. 1 also shows declining submission numbers – starting at 479 on the first falling to 129 on the last exercise sheet. The number of the exercise sheet and the number of submissions were found to be strongly negatively correlated (Spearman's $\rho(8) = -1.0$, $p < .000$, non-parametric test). The number of students who completed the reviews declines equally ($\rho(8) = -1.0$, $p < .000$).

The number of missing reviews and students who got no reviews first increase until the second resp. fourth sheet. When the exclusion rule took effect, these numbers continuously declines except for two (different) sheets; sheet 7 was due over the Christmas holidays. The number of students who got no reviews at all is between 5 to 21 % (arithmetic mean $\bar{x} = 14\,\%$). The majority of students always received two reviews ($\bar{x} = 73\,\%$). Interestingly, not all students wanted to see the received reviews ($\bar{x} = 42\,\%$ and $\bar{x} = 21\,\%$ of the students who completed the reviews) and from the students who got no reviews only a small fraction actually wanted to see reviews.

Starting from exercise sheet 8, all students were eligible to take part in the peer reviews again. For every submission, the students could mark a check box indicating whether they want to participate in the peer review for this assignment. This change in the procedure was conducted because on the one hand several students requested to be re-added again and on the other hand to see whether more students could be motivated to work on the assignments (again). For exercise sheet 8, 12 students voluntarily asked to be excluded and 13 previously excluded students requested to take part again; 8 of these finally provided peer reviews. For task 9 and 10, 12 students wanted to be excluded each, 15 respective 21 excluded students wanted to take part again, and of these 13 each delivered the peer reviews – no significant change to the prevailing pattern.

4.2 Analyzing the Reviews

In total there were 3975 peer reviews delivered by 451 distinct students. The median length of all reviews is 13 words[1] ($\bar{x} = 30$) or 87 characters ($\bar{x} = 203$). The number of words seems to increase over time (Spearman's $\rho(8) = .65$, $p = .041$ for the median number of words Wm; $\rho(8) = .75$, $p = .013$ for W$\bar{x}$, cf. Tab. 1). There are 237 reviews with ≥ 100 words (6 %) and 70 with ≥ 200 words (1.8 %).

To identify junk/bad feedback (such as containing single characters or many repeated characters) or feedback in which mainly code solutions were posted, the Shannon entropy was calculated to get a feeling on how much information is included in a feedback.[2] The median entropy is 4.25 bit ($\bar{x} = 4.017, min = 0, max = 5.257$). There are 28 reviews with an entropy of 0 (containing just a space, dot, etc.) that could be identified as junk (whereas 4 of these were for submissions without any solution attempt). When looking at the data sorted by entropy, there is a gap after these 28 junk reviews. Other reviews start with an entropy of 1 (containing an ASCII smiley ":)"; $n = 10$), or 1.5 containing one word (such as "good", "perfect" etc.). In the top-10 feedback with the highest entropy, 8 contained a significant amount of code relative to other explanations.

The median time for completing the reviews was 193 seconds ($\bar{x} = 403$ s) with a minimum of 6 and a maximum of 27540 s. There is a positive correlation between the variables length of the feedback and time needed (Spearman's $\rho(3973) = .68$, $p < .000$), also the avg. time spent on the reviews seems to increase over time ($\rho(8) = .90$, $p < .000$, cf. Tab. 1). The majority of students delivered their reviews roughly in the middle between the start and the deadline of the peer review assignments (i. e., for one week peer reviews: $m = 82.5$ h $= 3.4$ d, $\bar{x} = 85.6$ h before the deadline). The majority of reviews ($n = 2921$, 74 %) were delivered before the solution was discussed in the exercise groups the students registered for.

Students could also rate the received feedback with a thumb up or thumb down (not visible to students). Such a rating was given for 1664 reviews; 112 thumbs up and 1552 thumbs down. The longest negatively rated feedback has 75 words. The median of all negatively rated feedback is 7 words ($\bar{x} = 14.7$). For the thumb up rated feedback the shortest is a junk one containing just a space, the longest is 548 words with a median of 14 words ($\bar{x} = 29.4$). According to the Mann-Whitney UTest this difference is statistically significant ($U = 106596$, $p < .000$ two-sided). There also is a statistically significant difference between the writing time for positively ($m = 195$ s, $\bar{x} = 392$ s) and negatively ($m = 123$ s, $\bar{x} = 299$ s, $max = 3454$ s) rated feedback ($U = 101593$, $p = .003$).

4.3 Correctness of the Review Ratings

All Students had to rate their fellow students' submissions according to correctness and completeness on a Likert-scale from 1 to 4 (4 is best). This correctness rating comprised both, the syntactic and the functional correctness. For the evaluation, a contingency table and the χ^2 test of independence are used (cf. [3]). To map the integer values to a binary scale, a threshold of correctness=4 and completeness=4 is used to investigate whether the students' ratings correspondent to the unit tests' actual results. The syntactic correctness of Java submissions can be checked automatically by

[1] the feedback was split by white space ("\s+") and the resulting tokens were counted
[2] implementation based on https://stackoverflow.com/a/68031954

Table 1: Overview of the tasks, the peer reviews, and their characteristics (S: submissions, P: students with review assignments; C: completed reviews & self-assessments, R: review assignments, M: missing reviews, 2R: students who got 2 reviews, 0R: students who got zero reviews, WSR: students who wanted to view received reviews, S0R: students who actually saw no reviews, T: median/average time for the reviews in seconds, W: number of words, EP: students eligible to participate in peer reviews)

No. Sheet & Assignment/task	#S	#P	#C	#R	#M	#2R	#0R	#WSR	#S0R	Tm	T$\bar{x}$	Wm	W$\bar{x}$	#EP
1 Pseudo code	479	467	401	787	119	328	12	408	10	246	157	11	16	691
2 Algorithm properties	464	450	369	773	160	295	21	271	11	486	234	19	38	691
3 While loop	422	368	338	637	63	299	46	281	2	258	148	10	19	445
4 Collatz sequence	382	329	308	568	58	280	48	250	6	284	170	11	21	397
5 Count chars t,e,l in array	346	294	277	505	43	256	46	192	3	313	180	12	27	346
6 Class Fraction	295	250	220	425	48	205	45	151	3	520	254	16	44	311
7 Linked List	220	189	148	319	78	115	37	117	8	800	260	13	46	262
8 Binary searchtree	162	147	129	259	37	115	16	90	2	667	299	18	39	691
9 Collections & OO modelling	151	121	108	211	31	91	26	86	1	493	305	19	41	691
10 Custom Exceptions	129	114	75	196	68	46	24	66	7	589	331	21	43	691

running a compiler and simply checking the exit code. Checking the functional correctness automatically using unit tests requires a strict task specification, e. g. naming of methods and output formats. Please note that an invalid syntax also results in a failed unit test. Analyzed are the programming assignments excluding 7 and 8 (i. e., 3, 4, 5, 6, 9, and 10), because those two could not be tested automatically due to design freedoms in the solution. Tab. 2 shows an overview of the correct classifications per task and overall (each row except for task 10 shows a significant association with $p < .007$; the all tasks association: $\chi^2(1, n = 2231) = 271.24$, $p < .000$).

Table 2: Overview of the correctness/completeness classification quality of the students in the reviews and self-assessments (TP: true positives, FP: false positives, FN: false negatives, TN: true negatives, TPR: true positive rate (sensitivity), TNR: true negative rate (specificity), ACC: accuracy, SA: self-assessment)

Dataset	TN	FP	FN	TP	TPR	TNR	ACC
Task 3	87	88	57	342	.86	.50	.70
Task 4, strict	128	194	32	156	.83	.40	.56
Task 4, relaxed	116	194	32	168	.84	.37	.56
Task 5	115	112	22	213	.91	.51	.71
Task 6	165	123	25	64	.72	.57	.61
Task 9	78	83	3	16	.84	.48	.52
Task 10	59	69	0	0	-	.46	.46
All tasks	632	669	139	791	.85	.49	.64
Pre SA	294	457	60	515	.90	.39	.61
Post SA	354	397	61	514	.89	.47	.65

A total of 2231 peer reviews were delivered for the selected tasks whereas 1979 correspondent to syntactically correct and 930 to completely correct submissions. Across all tasks, students had an accuracy (sum of true positives and true negatives divided by the total number) of 64 %. Students seem to be better in classifying correct submissions as correct (TPR, sensitivity: 0.85) than in classifying incorrect submissions as incorrect (TNR, specificity: 0.46). This holds for all tasks whereas the sensitivity ranges from 0.72 to 0.91, the specificity from 0.4 and 0.57, and the accuracy from 0.46 to 0.71. No clear trends are visible over time and no significant correlations could be found. The overall false positive rate is 0.3 and the false negative rate 0.06. Here, statistically significant correlations could be found between time and false positive rate (Spearman's $\rho(4) = .82, p = .042$) and false negative rate ($\rho(4) = -.90$, $p = .042$).

As an exemplary case, assignment 4 (printing the Collatz sequence, e. g.: for number 5: 5, 16, 8, 4, 2, 1) is analyzed in more detail. For this assignment there are 382 submissions of which only 139 are correct (36 %). Here, a common problem was that the students often (n=132; 32 %) forgot to print the initial number – despite an example output on the exercise sheet. The strict correct version was correctly identified only in 156 reviews. In 114 reviews the incomplete version was rated with correctness=4 and completeness=4 (resulting in a false positive rate of 0.38); only in 12 reviews the incomplete version was identified as correctness=4 and completeness=3 (relaxed case in Tab. 2, only the sensitivity slightly improves and specificity slightly decreases). For the "All tasks" row in Tab. 2 and all analyses, the strict interpretation is used. A similar case, was found for assignment 3 where all uneven numbers from 1 to 10, and finally "Boom!" should be printed on the console. 40 students printed "BOOM!". As this was the first programming assignment and only a minor mistake, the unit test accepted both.

There is a positive correlation between the frequency of the delivery of correct submissions, the sensitivity (Spearman's $\rho(357) = .14$, $p = .007$), and the accuracy ($\rho(357) = .17$, $p = .001$). However, no statistically significant correlations between this frequency and the specificity, and between the accuracy and the number of delivered peer reviews could be found.

4.4 Self-Assessments

Seeing other solution strategies on coding tasks is quite often mentioned as very important by students (cf. Sect. 2 and 4.5). Therefore, the hypothesis that doing reviews and seeing other solutions has an impact on the view on the own submission (i. e., regarding correctness, elegance of the strategy, and coding style) is empirically investigated . First, the characterization of the dataset and the correctness of the self-assessments are presented.

The students performed 1603 complete self-assessments (pre and post) during the peer review for the 8 programming assignments – 31 self-assessments were not completed, i. e. only the pre self-assessment was delivered. For the programming tasks (except assignment 7 and 8, cf. Sect. 4.3) there were 1326 complete self-assessment delivered. The classification regarding the correctness=4 and completeness=4 by the students is shown at the bottom of Tab. 2 for both, the pre and post, self-assessments. In the pre self-assessments, the sensitivity is slightly higher than in the peer reviews (0.90 vs. 0.85) and the accuracy (0.64 vs 0.61) are slightly lower. For the specificity there is a larger difference (0.39 vs. 0.46). In the post self-assessments, 40 students (3 %) changed a false positive rating to a true negative rating. This results in a higher specificity (0.47 vs. 0.39 in the pre assessment). There is however one change from a true positive to a false negative. In general, the numbers get closer to the general peer review ratios.

Finally, the differences between the pre and post self-assessments are evaluated for all 8 programming assignments. The median time spent on the pre self-assessments was 32 seconds ($\bar{x} = 74$ seconds) and on the post self-assessments 14 seconds ($\bar{x} = 26$ seconds). The difference is statistically significant according to the Wilcoxon signed-rank test ($p < .000$). No trends over time for the pre and post assessment times could be observed. The results for the four dimensions of correctness, completeness, comprehensibility, and readability can be seen in Table 3. The differences in the ratings between the pre and the post self-assessment are only statistically significant for correctness and completeness ($p \leq .006$), however, both with a very small effect size. The completeness rating was changed 250 times (16 %) and the correctness rating 329 times (21 %) whereas it was lowered 150 resp. 190 times.

Table 3: Evaluation of the Self-Assessments (n=1603), p values according to Wilcoxon signed-rank test

Dimension	Pre $\bar{x}$	Pre m	Post $\bar{x}$	Post m	p
Correctness	3.54	4	3.50	4	.006
Completeness	3.63	4	3.60	4	.004
Comprehensibility	3.66	4	3.68	4	.173
Readability	3.67	4	3.66	4	.598

4.5 Questionnaires

In the questionnaire after the term, 243 students (male: 133, female: 82, median age 20-21) answered at least one item. The majority of students ($n = 164$) answered that they attended the lecture until the end. Only 79 students reported that they discontinued the course about halfway through the term (at chapter $m = 8, \bar{x} = 8.6$) when advanced topics were studied (chapter 7: dynamic data structures, chapter 8: sorting & complexity). Overall, the grade "good" was assigned by the students (1 very good, 6 inadequate; $m = 2, \bar{x} = 2.6$).

The closed questions regarding peer review are on a 5-point Likert-scale from 1 to 5 (1: "do not agree at all" to 5: "fully agree"). The majority of students agreed that the peer reviews of their fellow students were helpful ($n = 198, m = 4, \bar{x} = 3.7$) and particularly helpful for seeing new/alternative solutions ($n = 200, m = 4, \bar{x} = 4$). In general, the peer review was seen as meaningful ($n = 199, m = 4, \bar{x} = 3.8$), motivating ($n = 200, m = 4, \bar{x} = 3.5$), easy to do ($n = 295, m = 4, \bar{x} = 3.5$), and fun ($n = 294, m = 4, \bar{x} = 3.7$). When asked about the time needed for a peer review (self-assessment plus 2 peer reviews), 125 students answered with a median of 12.5 minutes ($\bar{x} = 17$ min.). The majority of students also preferred to be asked whether to take part in the peer review for each assignment instead of automatically taking part when submitting a solution and having strict exclusion rules ($n = 179, m = 4, \bar{x} = 3.6$).

The next part of the questionnaire was about the automatic tests. Here, a 4-point Likert-scale from 1 to 4 (1: never, 4: always) was used. The students answered that they used the automatic tests frequently ($n = 222, m = 3, \bar{x} = 3.0$) and found these frequently helpful ($n = 216, m = 3, \bar{x} = 2.8$). However, the automatic tests were only rated as rather motivating for them to work on the homework assignments ($n = 209, m = 2, \bar{x} = 2.3$). In a direct comparison, if only one of the approaches peer feedback or testing were available to the students (-2: peer review, -1: preferably peer review; 1: preferably automatic tests; 2: automatic tests), the majority prefers the peer review over the automatic tests ($n = 243, m = -1, \bar{x} = 0.1$).

In the following the tags and the associated frequencies of the qualitatively analyzed open-ended questions are presented. The reasons given for not participating are: "no time/other tasks were more important" (6x), "no perceived added value" (3x), and "too much pressure", "no feedback received", "correctness challenged", as well as "procrastination" (each 1x). The answers on the question "What did you like most about GATE and peer feedback?" are: "seeing other solutions" (30x), "peer review was helpful" (20x), "high usability of GATE" (structure, presentation of the tasks, ...) (9x), "peer review was motivating" (8x), "more interaction compared to other courses" (8x), "automatic tests" (6x), "positive sentiment in the feedback" (5x), "helpful to see possible errors" (4x), "like to help others" (2x), "giving feedback helps learning" (2x), and "comparison with others" (2x). Finally, the following improvements were named: "ask every time for PR participation" (8x), "sanctions [(exclusion from PR)] are too hard" (7x), "feedback was too short or uninformative" (5x), "get feedback for the given review" (4x), "unlimited tests" (4x), "discuss/ask questions in reviews" (3x), "availability of a sample solution for the review" (3x), and "editing peer reviews" (2x). Also, some general concerns were mentioned: "feedback was wrong" (2x), "too hard to give feedback" (2x), and "peer review should be abolished" (2x). Two further improvements were to automatically notify the students for the start of the peer review, and to revise the "hard to distinguish" ratings for completeness and correctness.

4.6 Exam

For the exam 635 students registered of which 416 students attended the exam and agreed their result to be correlated to their learning activities. 318 students (76 %) participated in the peer reviews at least once. The median score (max. 33) of the students with peer review is 22 and for the students without peer review 18 (overall: $m = 21, \bar{x} = 19.7$). This is a statistically significant difference according to the Mann-Whitney-UTest ($U = 21184$, $p < .000$ two-sided).

There also seem to be correlations between the number of completed peer reviews and the score (Spearman's $\rho(414) = .415$, $p < .000$) as well as between the number of delivered assignments and the score ($\rho(414) = .457$, $p < .000$).

5 DISCUSSION

The number of submissions decline for all tasks equally and are normal for university courses. Still, there are significantly more submissions for peer review and easier to solve multiple-choice tasks compared to normal programming tasks. Combined with the results from the questionnaire and the fact that not many students voluntarily signed out from the peer reviews starting from assignment 8 this indicates that students saw a clear advantage of the peer review over the other assignments for which only non-human feedback was available. The answers in the questionnaire on the preference of the peer review and liking the collaboration back this conclusion. However, the study was conducted during the pandemic and half of the semester was in lockdown. The rigid participation rules were criticized by some students but seem to have, in general, lowered the missing reviews rate and kept the participation up (RQ1). The change of the participation rule at the end of the semester, however, did not motivate significantly more students to participate (again).

The majority of students wanted to see their received peer reviews. It seems strange that not all students tried to view their received reviews. Maybe those students haven't found the link (it is the very same as for delivering the reviews) or are not interested.

To answer RQ2, the peer reviews were quantitatively analyzed. In general, the majority of the feedback was quite short (m=14 words) – this could also be observed in other research (e. g., [13] where tutor feedback was also found in this order). The time invested in the reviews seems to be quite short (m=3 min. vs. 12 min. reported in the questionnaire) but mostly not directly before the deadline. The number of words also seems be a good indicator for how valuable the feedback is perceived. However, the simple word count metric also shows greater numbers for source code and is therefore only a rough estimation. It might be combined with the Shannon entropy to also detect feedback in which mainly source code is posted. Still, intentionally rogue feedback such as a bad joke cannot be detected this way. Why the feedback length increases needs further research.

Students seem to be quite good in identifying correct submissions as correct (sensitivity 0.85) but not that good in classifying incorrect submissions as incorrect (specificity 0.46) – this contradicts other research (e. g., [13]). Except for task 10 there are more correct classifications compared to incorrect ones. It is hard to decide whether the students become better during the term. The sensitivity and specificity don't significantly change but the false negative rate decreases and the false positive rate increases. In comparison to an automated test, students often did not see or "punish" smaller mistakes in their ratings. Maybe the tasks get harder or more complex faster than the competencies increase. On the one hand, being more generous in ratings for minor mistakes which are not really significant for the exercised concept (e. g., a typo in the output, class, method, or package name) might be more motivating (also teaching assistants do this, cf. [31]). On the other hand, however, there are algorithms where subtle changes are significant, and students might not have understood the assignment, made the same mistake, or did not test the (review and their own) submissions. Providing automated test results to the students for the review could help. However, this might also lower the effort students put into the reviews. Still, a correlation that indicates that students who deliver more correct submissions have a higher accuracy was found.

A strict self-assessment process was used to evaluate RQ3. Students seem a bit more optimistic on the correctness and completeness of their own submitted solution in the pre self-assessments than in the post self-assessment. Peer reviewing other solutions seems to have an impact on how students perceive their own submission regarding correctness and completeness – for the 6 in more detail analyzed programming assignments the ratings seem to get closer to the truth. However, only a few students changed their rating and most students did not take much time to do the second self-assessment (median time: 14 s). Therefore, the impact seems to be rather low. It was also expected that seeing differently formatted source code has an impact on the own submission. This seems not to be the case. A further analysis of the coding style should be conducted. Additionally, this indicates that the ability to self-assess one's own solution attempt seems in general to be comparable to the assessment of other submissions. It might be interesting to see whether a similar effect can be seen if only a model solution is provided. However, this might contradict the often mentioned positive aspect of peer reviews to see different solution strategies.

A lot of different advantages have been named in the questionnaire that are also often intended by teachers (cf. [15], RQ4). Only individual students explicitly opposed peer reviews. Why students prefer peer review over automatic tests needs further investigation. In personal communication, getting feedback from a real person and also getting reasons why the code is not correct were named.

The correlation between the (peer review) activities and the exam score might just represent already engaged students.

6 CONCLUSIONS AND OUTLOOK

In this paper an approach for peer review in an introductory programming course is presented and analyzed. Working on the homework assignments was voluntary and peer review has shown to be a driver to motivate students to work on them. By using a strict participation rule for the peer reviews the missing review rate could be kept quite low. Doing reviews has shown to change students' view on their own solutions regarding the overall correctness. Overall, most students liked seeing other solutions, found the peer review helpful and motivating, and preferred it over automated tests.

For the future there are a number of aspects that should be evaluated: 1) To increase the quality of the reviews, the students could either be trained or a worked example should be presented. 2) Students request more flexibility on whether they want to participate in the reviews. Therefore, students could be allowed to decide for every assignment to take part in the reviews – in a combination with the rigid exclusion approach. This could lower the missing review rate but not providing a possible solution to others might harm their learning. 3) In this article there was a focus on a quantitative analysis. In following research the peer reviews could be sampled (due to the amount) and qualitatively evaluated (e. g., frequencies of hints, questions, rogue submissions/feedback). 4) Currently, there is a study undergoing in which students are instantly notified as soon as a new review is posted and can discuss with both reviewers. Reasons are that students explicitly requested this in the questionnaire, about 130 reviews contained questions and/or contact details, and discussions are a main aspect of peer reviews in industry [8, 1].

I thank all students who participated and also shared their data!

REFERENCES

[1] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *35th International Conference on Software Engineering (ICSE)*, 712–721. DOI: 10.1109/ICSE.2013.6606617.

[2] Ryan Baker, Jason Walonoski, Neil Heffernan, Ido Roll, Albert Corbett, and Kenneth Koedinger. 2008. Why students engage in "gaming the system" behavior in interactive learning environments. *Journal of Interactive Learning Research*, 19, 2, 185–224.

[3] Ralf Bender, Stefan Lange, and Andreas Ziegler. 2007. Wichtige signifikanztests. *Deutsche Medizinische Wochenschrift*, 132, e24–e25. DOI: 10.1055/s-2007-959034.

[4] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3, 2, 77–101. DOI: 10.1191/1478088706qp063oa.

[5] Tamaike Brown, Mourya Reddy Narasareddygari, Maninder Singh, and Gursimran Walia. 2019. Using Peer Code Review to Support Pedagogy in an Introductory Computer Programming Course. In *2019 IEEE Frontiers in Education Conference (FIE)*. IEEE, 1–7. DOI: 10.1109/FIE43999.2019.9028509.

[6] Kwangsu Cho and Christian D Schunn. 2007. Scaffolded writing and rewriting in the discipline: a web-based reciprocal peer review system. *Computers & Education*, 48, 3, 409–426.

[7] Rohan Roy Choudhury, HeZheng Yin, Joseph Moghadam, and Armando Fox. 2016. AutoStyle: toward coding style feedback at scale. In *Proceedings of the 19th ACM Conference on Computer Supported Cooperative Work and Social Computing Companion - CSCW '16 Companion*. ACM Press. DOI: 10.1145/2818052.2874315.

[8] Nicole Davila and Ingrid Nunes. 2021. A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software*, 177, (July 2021), 110951. DOI: 10.1016/j.jss.2021.110951.

[9] Michael de Raadt, David Lai, and Richard Watson. 2007. An evaluation of electronic individual peer assessment in an introductory programming course. In *Proceedings of the Seventh Baltic Sea Conference on Computing Education Research - Volume 88* (Koli Calling '07). Australian Computer Society, Inc., Koli National Park, Finland, 53–64. ISBN: 9781920682699.

[10] Dominik Dolezal, Alexandra Posekany, Christoph Roschger, Gottfried Koppensteiner, Renate Motschnig, and Robert Pucher. 2018. Person-centered learning using peer review method – an evaluation and a concept for student-centered classrooms. *International Journal of Engineering Pedagogy (iJEP)*, 8, 1, (Feb. 2018), 127–147. DOI: 10.3991/ijep.v8i1.8099.

[11] Stephanie J. Hanrahan and Geoff Isaacs. 2001. Assessing self- and peer-assessment: the students views. *Higher Education Research & Development*, 20, 1, (May 2001), 53–70. DOI: 10.1080/07294360123776.

[12] John Hattie and Helen Timperley. 2007. The power of feedback. *Review of Educational Research*, 77, 1, (Mar. 2007), 81–112. DOI: 10.3102/003465430298487.

[13] Niels Heller and Francois Bry. 2019. Organizing peer correction in tertiary stem education: an approach and its evaluation. *International Journal of Engineering Pedagogy (iJEP)*, 9, 4, 16–32.

[14] Christopher Hundhausen, Anukrati Agrawal, Dana Fairbrother, and Michael Trevisan. 2009. Integrating pedagogical code reviews into a cs 1 course: an empirical study. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (SIGCSE '09). Chattanooga, TN, USA, 291–295. DOI: 10.1145/1508865.1508972.

[15] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. 2020. A review of peer code review in higher education. *ACM Transactions on Computing Education*, 20, 3, (Sept. 2020), 1–25. DOI: 10.1145/3403935.

[16] Theresia Devi Indriasari, Andrew Luxton-Reilly, and Paul Denny. 2020. Gamification of student peer review in education: a systematic literature review. *Education and Information Technologies*, 25, 6, (May 2020), 5205–5234. DOI: 10.1007/s10639-020-10228-x.

[17] Amandeep Kaur and Robin Kumar. 2015. Comparative analysis of parametric and non-parametric tests. *Journal of computer and mathematical sciences*, 6, 6, 336–342.

[18] C.F. Kemerer and M.C. Paulk. 2009. The impact of design and code reviews on software quality: an empirical study based on PSP data. *IEEE Transactions on Software Engineering*, 35, 4, (July 2009), 534–550. DOI: 10.1109/tse.2009.27.

[19] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2018. A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, 19, 1, Article Article 3, (Sept. 2018), 43 pages. DOI: 10.1145/3231711.

[20] Christopher Laß, Stephan Krusche, Nadine von Frankenberg, and Bernd Brügge. 2019. Stager: simplifying the manual assessment of programming exercises. In *Proc. SEUH* (CEUR Workshop Proceedings). Vol. 2358. CEUR-WS, 34–43. http://ceur-ws.org/Vol-2358/.

[21] Xiao Liu and Gyun Woo. 2020. Applying code quality detection in online programming judge. In *Proceedings of the 2020 5th International Conference on Intelligent Information Technology*. ACM, (Feb. 2020). DOI: 10.1145/3385209.3385226.

[22] Andrew Luxton-Reilly et al. 2018. Introductory programming: a systematic literature review. In *Proc. ITiCSE*, 55–106. ISBN: 9781450362238. DOI: 10.1145/3293881.3295779.

[23] David Nicol. 2010. From monologue to dialogue: improving written feedback processes in mass higher education. *Assessment & Evaluation in Higher Education*, 35, 5, 501–517. DOI: 10.1080/02602931003786559.

[24] David Nicol, Avril Thomson, and Caroline Breslin. 2013. Rethinking feedback practices in higher education: a peer review perspective. *Assessment & Evaluation in Higher Education*, 39, 1, (May 2013), 102–122. DOI: 10.1080/02602938.2013.795518.

[25] Lorelli S. Nowell, Jill M. Norris, Deborah E. White, and Nancy J. Moules. 2017. Thematic analysis: striving to meet the trustworthiness criteria. *International Journal of Qualitative Methods*, 16, 1. DOI: 10.1177/1609406917733847.

[26] Ryan Rybarczyk and Lingma Acheson. 2019. Interactive peer-led code reviews in CS2 curricula. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. ACM, (Feb. 2019). DOI: 10.1145/3287324.3287442.

[27] Joanna Smith, Joe Tessler, Elliot Kramer, and Calvin Lin. 2012. Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on International computing education research*. ACM, (Sept. 2012). DOI: 10.1145/2361276.2361295.

[28] Harald Søndergaard and Raoul A. Mulder. 2012. Collaborative learning through formative peer review: pedagogy, programs and potential. *Computer Science Education*, 22, 4, 343–367. DOI: 10.1080/08993408.2012.728041.

[29] Saikrishna Sripada, Y Raghu Reddy, and Ashish Sureka. 2015. In support of peer code review and inspection in an undergraduate software engineering course. In *2015 IEEE 28th conference on software engineering education and training*. IEEE, 3–6. DOI: 10.1109/CSEET.2015.8.

[30] Sven Strickroth and Florian Holzinger. 2022. Supporting the semi-automatic feedback provisioning on programming assignments. In *Methodologies and Intelligent Systems for Technology Enhanced Learning, 12th International Conference*. Marco Temperini, Vittorio Scarano, Ivana Marenzi, Milos Kravcik, Rosa Popescu Elviraand Lanzilotti, Rosella Gennari, Fernando De La Prieta, Tania Di Mascio, and Pierpaolo Vittorini, (Eds.) Springer International Publishing, Cham, 13–19. ISBN: 978-3-031-20616-0. DOI: 10.1007/978-3-031-20617-7_3.

[31] Sven Strickroth, Hannes Olivier, and Niels Pinkwart. 2011. Das GATE-System: Qualitätssteigerung durch Selbsttests für Studenten bei der Onlineabgabe von Übungsaufgaben? In *Tagungsband der 9. e-Learning Fachtagung Informatik (DeLFI)* (GI Lecture Notes in Informatics). Holger Rohland, Andrea Kienle, and Steffen Friedrich, (Eds.) Gesellschaft für Informatik e.V., Bonn, Germany, 115–126. https://dl.gi.de/handle/20.500.12116/4740.

[32] Sven Strickroth and Michael Striewe. 2022. Building a corpus of task-based grading and feedback systems for learning and teaching programming. *International Journal of Engineering Pedagogy (iJEP)*, 12, 5, (Nov. 2022), 26–41. DOI: 10.3991/ijep.v12i5.31283.

[33] Qing Sun, Ji Wu, Wenge Rong, and Wenbo Liu. 2019. Formative assessment of programming language learning based on peer code review: Implementation and experience report. *Tsinghua Science and Technology*, 24, 4, 423–434. DOI: 10.26599/TST.2018.9010109.

[34] Keith Topping. 1998. Peer assessment between students in colleges and universities. *Review of Educational Research*, 68, 3, (Sept. 1998), 249–276. DOI: 10.3102/00346543068003249.

[35] Scott Alexander Turner, Manuel A. Pérez-Quiñones, and Stephen H. Edwards. 2018. Peer review in CS2. *ACM Transactions on Computing Education*, 18, 3, (Sept. 2018), 1–37. DOI: 10.1145/3152715.

[36] Yanqing Wang, LI Yijun, Michael Collins, and Peijie Liu. 2008. Process improvement of peer code review and behavior analysis of its participants. In *Proceedings of the 39th SIGCSE technical symposium on Computer science education*. ACM, (Mar. 2008). DOI: 10.1145/1352135.1352171.

[37] Wang Yanqing, Li Hang, Sun Yanan, Jiang Yu, and Yu Jie. 2011. Learning outcomes of programming language courses based on peer code review model. In *2011 6th International Conference on Computer Science & Education (ICCSE)*. IEEE, 751–754. DOI: 10.1109/ICCSE.2011.6028746.

[38] Andreas Zeller. 2000. Making students read and review code. In *Proceedings of the 5th Annual SIGCSE/SIGCUE ITiCSEconference on Innovation and Technology in Computer Science Education* (ITiCSE '00). Association for Computing Machinery, Helsinki, Finland, 89–92. ISBN: 1581132077. DOI: 10.1145/343048.343090.