Qualitative Analysis of Peer Reviews of a Large Introductory Programming Course

Sven Strickroth and Imen Azaiz

LMU Munich, Munich, Germany

ARTICLE HISTORY
Compiled January 15, 2025

ABSTRACT
Background and Context. Getting timely feedback is important for learning. However, providing individual feedback is a problem in large courses. Peer code review can address this issue and has shown to offer various advantages such as enhanced collaboration among students, improved coding skills, and seeing and criticizing different solution strategies.
Objectives. The goal is to gain a comprehensive understanding of the contents of voluntary peer reviews and students' review skills in a first-semester university-based introductory programming course with about 900 enrolled students and no special support or training for conducting reviews.
Method. A qualitative analysis was conducted on 215 randomly sampled peer reviews as well as the associated code submissions. Qualitative factors such as frequencies of corrections, coding tips, questions, "empty" submissions/feedback, and the sentiment are analyzed. Furthermore, errors in the submissions and mentioned errors in the reviews are investigated exploratively.
Findings. The analysis results indicate that, in general, students are better in detecting incorrect solutions as incorrect than correct submissions as correct. The feedback is neutral to positive, contains a lot of praise, but is rather short and uncertainty is expressed quite often. Students seem to be very nice to each other. Submissions without intensive solution attempts and "empty" reviews are quite rare. Students can correct errors and provide coding tips, however, often do not see subtle errors such as partly incorrect algorithms, typos in method names, or output formatting errors.
Implications. The results help to train students to write better reviews and to inform educators on how to provide better instruction or support for peer review.

KEYWORDS
peer code review; peer teaching; peer instruction; formative feedback; programming education; qualitative analysis

1. Introduction

Peer reviews are widely used in industry, open-source projects, and also as a collaborative teaching activity in (programming) education (Davila & Nunes, 2021; Indriasari, Luxton-Reilly, & Denny, 2020b). As a teaching method, peer review is an activity in which students with similar competencies evaluate their fellow students' submissions and provide feedback. This method addresses both technical and soft skills, such as learning to collaborate, giving and receiving criticism, and seeing different solution strategies

CONTACT Sven Strickroth. Email: sven.strickroth@ifi.lmu.de

(Indriasari et al., 2020b; Nicol, 2010). Both, the reviewer and reviewees are expected to benefit.

Peer review is particularly helpful for classes that are attended by hundreds of students because social interaction, discussion, and feedback become rather limited in such courses (Strickroth & Bry, 2022). Providing timely and personalized feedback to all students is difficult due to high teacher-to-student ratios (Strickroth & Bry, 2022; Sun, Wu, Rong, & Liu, 2019; Søndergaard & Mulder, 2012) despite the enormous importance of feedback to learning success (Hattie & Timperley, 2007). To support such scenarios, many (e-assessment) systems have been developed (Keuning, Jeuring, & Heeren, 2018; Strickroth & Striewe, 2022) to provide automatic feedback to students or pre-corrections for teaching assistants (TA, cf. Strickroth & Holzinger, 2022). However, they often only focus on functional correctness and neglect qualitative aspects such as coding style, creativity, code quality etc. or the provided coding style feedback is not sufficient for students (Choudhury, Yin, Moghadam, & Fox, 2016; Keuning et al., 2018; Liu & Woo, 2020). A significant advantage of peer reviews over the automatic evaluation of students' submissions is that, additionally to the above mentioned competencies, no tests need to be developed and it can be used to provide (holistic) feedback for incomplete submissions, coding style, algorithm design etc., and even for ill-defined tasks such as modeling (cf. Le, Loll, & Pinkwart, 2013).

Despite all the benefits that have been reported for peer review in several studies, there are also barriers such as a lack of knowledge, motivation, interest, or ability to produce accurate and useful feedback (Indriasari, Luxton-Reilly, & Denny, 2021). To be able to support students in providing good reviews and to optimize peer review processes, it is necessary to get further insights on the contents of the reviews and the review abilities of the students. This paper presents an in-depth qualitative analysis of peer reviews conducted in a large introductory programming class, and it answers the following research questions:

- RQ1: How can the submitted peer reviews be characterized qualitatively (regarding correctness, frequencies and type of hints, questions, "empty" submissions/feedback, praise, sentiment etc.)?
- RQ2: How do the characteristics change over the semester?
- RQ3: What review skills of the students can be identified?

The contributions of the paper are the developed coding book, a characterization of the submitted code submissions as well as the submitted peer reviews in our scenario, insights into trends over the semester, a classification of the peer reviews using Narciss (2008) feedback categories, and an exploratory analysis of weaknesses and strengths of students' review skills. These contribute to the understanding of student peer code review and provide educators insights into the characteristics of peer reviews, which can aid them in optimizing peer review processes and providing (targeted) support to students.

The paper starts with outlining the related research on automatic grading and peer reviews. It then describes the evaluation setting and presents the methodology and results. The paper closes with a discussion, a summary, and an outlook.

2. Related Research

In general, peer review allows students to obtain timely and extensive feedback as well as exchanging ideas (Dolezal et al., 2018; Hanrahan & Isaacs, 2001; Indriasari et al.,

2020b; Søndergaard & Mulder, 2012). Positive impacts on learning such as an increased time spent on a subject due to the reviews and self-reflection have been reported (Nicol, Thomson, & Breslin, 2013; Topping, 1998). Also, multiple peer reviews can get close to or be even better than a single (expert) review (Cho & Schunn, 2007; Hamer, Purchase, Denny, & Luxton-Reilly, 2009; Reily, Finnerty, & Terveen, 2009).

A recent systematic literature review on peer code review in higher education by Indriasari et al. (2020b) compiles benefits, barriers, and challenges associated with the use of student peer code review: On the one hand, there are many studies where the students were very motivated to conduct peer code review (e.g. de Raadt, Lai, & Watson, 2007; Rybarczyk & Acheson, 2019; Smith, Tessler, Kramer, & Lin, 2012) and where the students also reported the experience as very valuable for collaboration and learning (Indriasari et al., 2020b). There are reports that the quality of the peer reviews, the code, and the students' ability to detect errors can improve over time (Brown, Narasareddygari, Singh, & Walia, 2019; Hundhausen, Agrawal, Fairbrother, & Trevisan, 2009; Sripada, Reddy, & Sureka, 2015): In a study by Hundhausen et al. (2009), students were able to correct low-level syntax and style issues as well as higher-level design and implementation issues in moderated peer code review sessions. On the other hand, a lack of engagement/motivation is often identified as a major problem (e.g. Heller & Bry, 2019; Indriasari, Luxton-Reilly, & Denny, 2020a; Sripada et al., 2015; Wang, Yijun, Collins, & Liu, 2008). Also, the length of the reviews can be quite short (e.g. Strickroth, 2023, for a large course: median length 13 words; Heller & Bry, 2019: median length 14 words; or Hamer, Purchase, Luxton-Reilly, & Denny, 2015: mean length 25.25 tokens) or the assessment quality can be an issue (Strickroth, 2023: accuracy 64%; Heller & Bry, 2019: 22% reviews are partly incorrect). Reported reasons, apart from motivation, for these issues are students rushing through the reviews (Sripada et al., 2015), an actual or perceived lack of student understanding (Wang et al., 2008), or too little time is allocated for the review (e.g. Stalhane, Kutay, Al-Kilidar, & Jeffery, 2004). Lastly, there are also reports of no improvements of students' review skills and the reviews over time (e.g. Turner, Pérez-Quiñones, & Edwards, 2018). Overall, Indriasari et al. (2020b) found that most studies are only based on self-reports in surveys and interviews in courses with less than 200 students. Only few studies investigated the contents of the reviews, hence, they do not provide any insight into what is actually written in the reviews. Indriasari et al. (2020b) also note that almost one-third of the reviewed studies did not provide detailed information on the course level. Detailed information on the provided type/amount of training offered to students is also often missing.

There are studies investigating the content of peer review qualitatively in programming related courses: Politz, Collard, Guha, Fisler, and Krishnamurthi (2016) used peer reviews in the context of developing test cases. They analyzed the content of the reviews of different courses and developed a coding book comprising seven categories (*Identified (Potential) Mistake*, *Discussed or Disputed Problem Specification*, *Suggested Additional Test(s)*, *Positive*, *Hostile/Negative*, *Coding or Style Tip*, and *Reviewer Gained/Confirmed Understanding*). The most and second most common features identified are *Suggested Additional Test(s)* and *Identified (Potential) Mistake*. Furthermore, they identified 16–45% cursory (largely content-free but non-empty) reviews which they categorized as *LGTM* (Looks good to me). It is unclear whether these results also apply to peer code review. Also notable is a study comparing peer reviews to tutor feedback in terms of quantity (tutor comments are longer, more specific and use more technical vocabulary), content dimensions (such as positive/negative comments; no significant difference), and other personal characteristics such as gender (Hamer et

al., 2015, 2009). Indriasari, Denny, Lottridge, and Luxton-Reilly (2022) investigated a gamification approach to nudge students to write better code reviews and found significant improvements in quantity and quality. Their system requested students to write comments for seven pre-defined categories (*Variable names*, *Comments*, *Layout*, *Expressions*, *Control flow*, *Decompositions*, and *General*). In their evaluation, Indriasari et al. used a qualitative approach to categorize the peer reviews into an adapted version of the taxonomy of Hamer et al. (2015) with ten categories. Their main categorization is based on two dimensions (resulting in eight categories): first dimension, *general* and *specific*, and second dimension, *positive*, *negative*, *neutral* and *advice*. Pointing out errors was categorized as *negative*. This, however, may not reflect the actual sentiment in the comment. Additional to the two dimensions, the last two categories are *personal voice* that includes encouragement and *off-topic*. Most feedback was categorized as *specific positive*, i.e. the comments "highlight specific aspects that meet the criteria, or provide positive feedback about specific elements" (Indriasari et al., 2022, p. 469). Their categories are rather coarse grained (many aspects that we found such as "empty" comments, empathy, uncertainty, or providing solutions were not categorized) and no connection is made from a review to the corresponding submission.

Studies by Cho, Schunn, and Charney (2006) and Cho, Chung, King, and Schunn (2008) on peer reviews in academic writing examined types of comments (such as suggestions for specific or unspecific changes, praise, criticism) and found that student comments contain more praise and fewer specific suggestions compared to an expert. Code reviews in open source projects and industry have been analyzed qualitatively and also the sentiment in the reviews (Davila & Nunes, 2021): Key points are to find defects, provide feedback, improve the code, discuss alternative approaches, and knowledge sharing (Bacchelli & Bird, 2013; Davila & Nunes, 2021). Questions are of particular importance in this process as most of the suggestions are made by raising questions (Bacchelli & Bird, 2013; Davila & Nunes, 2021). Core developers are neutral when commenting a review compared to more positive/negative comments by other collaborators (Asri, Kerzazi, Uddin, Khomh, & Idrissi, 2019). There is a research gap in how these aspects are included in student peer code reviews.

Narciss (2008) investigated feedback types and distinguished two categories: simple (e.g. knowledge of result/response, knowledge of performance) and elaborated feedback (e.g. knowledge about mistakes and knowledge about how to proceed). Particularly, elaborated feedback is often hard to achieve in automated systems (Jeuring et al., 2022; Keuning et al., 2018). There is also evidence suggesting that peer review is perceived as superior to automated tests for this reason (Strickroth, 2023). The authors are not aware of any analysis of peer code reviews according to this categorization.

In summary, peer review offers great pedagogical benefits. However, there are contradictory results reported and there is no in-depth systematic qualitative analysis of the peer code review content to find general characteristics known to the authors (only papers analyzing (Chat)GPT feedback qualitatively, cf. Azaiz, Deckart, & Strickroth, 2023; Azaiz, Kiesler, & Strickroth, 2024; Kiesler, Lohr, & Keuning, 2023). Indriasari et al. (2020b) request more empirical research, particularly in larger courses, in their review. Hence, this paper identifies characteristics as well as weaknesses and strengths of the feedback and students' review skills so that review processes can be optimized and targeted support can be provided. There is no qualitative analysis which analyzes both the submissions and the reviews or classifies the reviews according to Narciss's feedback categories.

3. Setting

The peer reviews were collected in an introductory programming course on Java at LMU Munich, Germany in winter term 2021/22. About 900 students with computer science as a major or minor were enrolled in that course. Working on weekly homework assignments as well as taking part in the peer review was voluntary. Apart from the peer review, there was no individual correction of the students' submissions. Nevertheless, students were able to get automatically generated feedback from the GATE system (Strickroth & Holzinger, 2022; Strickroth, Olivier, & Pinkwart, 2011) for the non-peer-reviewed submissions, and there were weekly exercise sessions distributed over the whole week led by 18 student teaching assistants in which solutions were discussed in the week after the submission deadline in the GATE system. The teaching assistants were not involved in the peer review, reviews were not discussed, and there was no special training for the students for providing feedback. At the end of the term, there was a written exam that constituted the final grade for the course. Peer review was conducted for one assignment on each of the ten exercise sheets (to not overwhelm the students, cf. Section 2) and was expected to be done as part of the homework assignment by the students. The exercise sheets were distributed and due on Thursday, so there was an overlap of the exercise sessions with the peer review for organizational reasons; 74% of the students delivered their peer reviews before attending their exercise session. The first two exercise sheets did not contain any programming tasks, but a peer review was conducted to get into the process. Hence, there were 8 peer code reviews. Each week students completed one exercise sheet and wrote a peer review for the previous week's exercise sheet as homework. The peer review was managed using an adapted version of the GATE system.

The overall peer review process was as follows: All students who submitted their solution for an assignment with peer review on the first seven exercise sheets automatically participated in the peer review process as long as they have not skipped two assignments or did not deliver peer reviews twice (strict exclusion rule, cf. Strickroth, 2023). Starting with the eighth sheet, all students (also the excluded ones) were explicitly asked whether they wanted to participate in the peer review for the next round — this change in the process was introduced for another (qualitative) evaluation (cf. Strickroth, 2023). Each week, participating students had to deliver two reviews for two randomly assigned submissions (i.e. one review for each submission).

For each review, the students had to assess their fellow student's submission on a 4-point Likert scale (4=best) according to correctness, completeness, readability (referring to style), and comprehensibility of the approach to the solution, and had to write an open-ended comment. The prompt was "Please give your fellow students (positive) feedback and suggestions for possible improvements:" (translated to English) and remained unchanged throughout the whole semester. For the review, the submitted source code was displayed with syntax highlighting in an iframe. Students could copy the code (mostly consisting of a single file) easily but there was no download link provided. The reviews were entered separately from the submitted source code in a text area. After the deadline, the reviews became visible to all students at once — even if they did not deliver their assigned reviews. Each student could only see the reviews s/he gave and received. The author of the code and the author of the reviews remained anonymous (unless students explicitly revealed their identity).

All students were explicitly asked whether their anonymized submissions and peer reviews can be used for research. Consent was fully voluntary without any negative consequences in case they did not agree. Based on local regulations, this procedure is

sufficient. The whole dataset consists of about 700 students who agreed and delivered 3050 submissions (i.e. 69% of the whole dataset) and had 4680 peer reviews assignments (58% of all peer review assignments). There are 533 students who had at least one peer review assignment of which 451 students delivered at least one peer review overall.

4. Method

During the semester, 2109 of the review assignments were completed for the programming assignments (completion ratio 85%) by 335 distinct students. Of these, 215 peer reviews (about 10%) were randomly sampled and manually qualitatively analyzed. The number of about 10% was chosen arbitrarily to keep the workload manageable. Mayring's thematic analysis technique (Braun & Clarke, 2006; Mayring, 2001) was used for the deductive-inductive category-building process. As a first step, coding units were defined for the qualitative analysis. Students provided their written feedback as one text that also often is quite short. Therefore, it was decided to use the whole feedback text as one coding unit. As the feedback can exhibit (intertwined) characteristics, multiple codes may be assigned to one coding unit.

Ten categories were chosen as an initial set to be able to identify aspects of interest that are not present in the dataset. The chosen aspects are inspired by related work (Azaiz et al., 2023, 2024; Cho et al., 2008, 2006; Davila & Nunes, 2021) and are (marked with "*" in Table 3): explicit correctness assessment, style assessment, praise, general/specific coding tips, general/specific corrections, questions, solution, and "emptyness".

Starting from the initial set of ten categories all reviews were analyzed and tagged whether they contain specific characteristics in the original language (i.e. German). When interesting new aspects were found, such as "I feel your pain" (empathy), or when refinements to the categories were needed such as different types of correctness assessments (i.e. very broad statements that encompass correctness such as "everything's fine"), new categories were iteratively inductively introduced and already tagged submissions re-checked. For each category, a definition with anchor examples was established and added to the coding book.

At the same time, the associated submissions to the peer reviews were also manually analyzed. There are 208 distinct code submissions, i.e. there are two reviews for seven submissions. The submissions were investigated whether they represent an intensive solution attempt (cf. Section 5.1), contain Java source code comments, are syntactically correct (based on the OpenJDK 11 compiler), or fully correct according to the task specification (unit tests and also manually) — see Table 1 for an overview. Again, the whole code submission was used a one coding unit. Comments were analyzed to detect possible interactions between submitters and reviewers. In addition to the coding of the reviews, the students' correctness statements and corrections were evaluated for their actual correctness, and whether provided coding tips introduce new errors. The types of errors found and not found, as well as suggested coding tips, are finally summarized in an exploratory manner.

To ensure objectivity and clearness of the coding book, about 12% (n=25) of the data was also independently coded by a second researcher. Cohen's κ (Cohen, 1960) was calculated for each of the categories — with a minimum of .759 for the category *praise* which is considered substantial (Landis & Koch, 1977; McHugh, 2012). Overall, there is a (very) high agreement with maximum of 1 to 2 disagreements per category. The κ values for the categories are as follows: *quality assessment* (overall) .816, *style*

Table 1. Coding book used for the analysis of the submissions

Category	Definition	Examples
Syntactically correct	Checked using OpenJDK 11 compiler	n/a
Fully correct	Checked against the task specification (unit tests and also manually)	n/a
No intensive solution attempt	Almost empty solution or no sensible solution attempt.	unmodified template, template with only some comment(s), empty class definition
Code comments	Comments on the (structure of the) code	"`// Constructor`", "`/* case n uneven*/`", "`// returns the result of ...`"
	Question or interaction with a reviewer	"gave up after long puzzling", "don't know what to do", "please post your solution"

assessment .781, *praise* .759, *(general and specific) coding tips* .865, *correction* 1, *specific correction* .865, *uncertainty* 1, *negativity* 1, *emoji* 1, *thanks* 1, *motivation* 1, *references* .779, and *emptyness* 1; all these are considered a substantial (.61 to .8) to almost perfect agreement (> .81) (Landis & Koch, 1977; McHugh, 2012). No κ value could be calculated for *Formatting hint*, *learning*, *highlighting*, *empathy*, *question*, *invalid term*, *testing*, *contact*, and *solution*, because these categories were not present resp. not identified by both raters in the dataset used for the inter-rater agreement evaluation. Implications are discussed in Section 7. All cases of disagreement identified in the inter-rater agreement evaluation as well as further unclear cases during the analysis were discussed, and if necessary the coding book was updated as well as a re-coding of relevant reviews (Mayring, 2001).

In the results section, the found categories are also related to the feedback categorization of Narciss (2008) where appropriate. All quotes from the peer reviews are translated from German to English for this paper.

The quantitative assessment (i.e. the scores for correctness and completeness) in the peer review process was used for another quantitative study (cf. Strickroth, 2023). In this analysis, it is only used to see whether students provided reasons when they did not assign a full correctness score.

5. Results

A total of 23 categories (cf. Table 3) were developed from the analysis of the peer reviews; examples for all initial categories were found. The first category *quality assessment* mainly deals with correctness assessment. Hence, a statement such as "perfectly readable" would not be tagged as *explicit general quality assessment statement that encompasses but is not limited to correctness*, whereas "good/nice solution" and "looks good" would be as these statements also encompass correctness to a specific degree (at least in the original language). Notable are five peer reviews regarding their categorization in *explicit statement judging the correctness or indicating an error, problem, or missing aspect in the submission under review* as they led to intensive discussion: First, there are three reviews that contain a correction such as "you must distinguish between lowercase and uppercase letters" which are included in this category. However, one review stating "xy is missing and thus I could not check your solution" and one review stating a possible correction as a question ("Doesn't line 18 have to be 'else if'?") are not. The category *coding tips* also encompasses *specific coding tips*, the same applies

to *correction* that also encompasses *specific correction*. The *negativity* category does not include reviews that point out errors in a neutral way, but is only used when the comment has a negative tone.

First, the code submissions are characterized, then the reviews. After that the correctness, errors, corrections, and coding tips are discussed.

5.1. General Characteristics of the Submissions

According to the specification of the assignments, 65 out of the 208 submissions are fully correct (tested using unit tests and double checked manually). The manual inspection for full correctness was necessary, because there is one submission for assignment 7 that passes the unit test but internally uses an *ArrayList* which violates the task specification and that would not have been detected otherwise. Based on the OpenJDK 11 compiler, 38 submissions are syntactically incorrect. Table 2 provides an overview of the assignments and submissions. Except for assignment 7 and assignment 8, where a singly linked list respective a binary tree should be implemented, the syntactic correctness is always higher than 83%. For these two assignments 32–41% of the submissions are detected as syntactically incorrect, because a central class (e.g. *ListEntry/TreeNode*) is missing. No clear trend over time is observable. However, for the full compliance to the assignment specification there seems to be a clear negative trend: In the first assignment the majority of submissions are completely correct (57%) and for the last three assignments exactly one (details in Section 5.3).

There are 12 submissions (about 6%) that do not seem to indicate an intensive solution attempt. All originate from different students. The first such submission was found for assignment 5. Overall, five submissions only contain the Java class definition or the provided template without any visible solution attempt.

One submitted file for assignment 5 contains the Java comment "I'm so sorry. I didn't save my work and this is all that is left... :((". Overall, seven submissions are mostly incomplete but contain some code trying to solve parts of the assignment — five submission contain a comment indicating some confusion, e.g. "No idea I am confused :(", "Sry could not solve the task", or "[I] gave up after long puzzling". Furthermore, there are two submissions that contain a plain text (i.e. not a Java comment and, thus, resulting in a syntax error) similar to "no idea :(am confused". Overall, questions in comments or other interactions with the reviewer, such as asking for posting a solution in the review, could be found in 7 submissions; interestingly, only for three assignments in the middle of the semester.

The compiler reported failures for 7 otherwise correct submissions because of encoding issues (umlauts were used e.g. in a comment, but the file was not UTF-8 encoded) for assignments 5 and 8. Furthermore, there are 10 submissions that use a (wrong) package name, 3 submissions in which a wrong class name is used, 11 submissions with a typo in a method name, and 7 submissions with missing methods or modified method signatures that cause automatic tests to fail early. These errors are not limited to early assignments and can be found across nearly all assignments. Notably, wrong class names could only be found for the last two assignments.

In 42 submissions (20%) there are Java code comments included that are written by students and commenting on the code or its structure. With the exception of the fifth and the last assignment, the percentage of submissions with such code comments seems to increase over time. Finally, there are 7 submissions that contain commented out Java code.

Table 2. Overview of the tasks and the characteristics of the submissions (S: submissions, NS: no intensive solution attempt, SC: syntactically correct, FC: fully correct, C: code/interaction comments)

No.	Assignment Topic	#S	#NS	#SC (%)	#FC (%)	#C (%)
3	While loop, print uneven numbers 1–9 and "Boom!" at end	42	0	40 (95)	24 (57)	4 (10)/0
4	Collatz sequence (using a loop, e.g. 1 or 5, 16, 8, 4, 2, 1)	37	0	34 (92)	9 (24)	4 (11)/0
5	Count frequency of chars (t|T), (e|E), (l|L) in a char array	33	1	28 (85)	16 (49)	3 (9)/0
6	Class Fraction (Constructor, Add, Multiply, asDouble, toString)	29	4	25 (86)	7 (24)	8 (26)/4
7	Singly Linked List (implementing a given interface)	22	1	15 (68)	8 (36)	8 (28)/2
8	Binary search tree (implementation and in-order recursive traversal)	17	2	10 (59)	0 (0)	6 (35)/1
9	Collections & OO modelling (Exam with questions in lists)	16	3	14 (88)	1 (6)	7 (44)/0
10	Custom Exceptions (Book and ISBN10/13 with validity check)	12	1	10 (83)	0 (0)	2 (17)/0
	Overall	208	12	176 (85)	65 (31)	42 (20)/7

5.2. General Characteristics of the Reviews

The mean length (in words) of the randomly sampled reviews is 31, the median is 11. More specifically, for incorrect submissions the mean length is 39 and the median 11 and for fully correct submissions the mean length is 15 and the median 8. Tables 4 and 5 provide an overview of the review characteristics (coding book see Table 3). Further less frequent characteristics, such as questions in the reviews, are not included in Tables 4 and 5 but are discussed in the text only.

Overall, 46% of the reviews contain an *explicit statement judging the correctness or an indication of an error* (cf. knowledge of result/response feedback category of Narciss, 2008). This percentage ranges between 36 and 63 for all tasks with the minimum for assignments 4 and maximum for assignment 6 (no visible trend). Additionally, several general correctness statements such as "perfect" and "very good" could be identified but seem to decrease over time. Combined, 83% of all reviews contain one of the two forms of quality judgment that encompasses correctness. Interestingly, there are 11 reviews with correctness and completeness scores of less than 4 without any indication of an error in the review text: Most of these reviews state something like "looks correct". Only in one case it was noted that the reviewer could not compile the (actually correct) submission.

An assessment of coding style, clarity of the solution, understandability, or readability of code can be found in 22% of the reviews. The percentage seems to be quite stable around 25% for all assignments with an exception for assignments 7 and 10 with only 14 respective 8%. The maximum is 29% for assignment 8.

Overall, the sentiment in the reviews can be considered neutral to positive. More than half of the reviews (n=123) contain some kind of praise or positive valuation. In 14 reviews (about 7%) motivational phrases such as "keep it up" could be identified. There are 50 reviews that contain an emoji — only positive emojis were found. No trend over time is visible: there are emojis in 20–35% of the reviews except for the last two assignments (13% and 0%). Three reviews contain the word "thanks" (or synonyms). Furthermore, there are 6 reviews that indicate empathy with their fellow students such as "Happens.Better Luck next time!" for the submission where the student commented that s/he lost his/her work. Statements with a negative sentiment such as "unfortunately not very well written code", "completely cluttered [code]", and

Table 3. Coding book used for the qualitative peer review analysis (* initial set)

Category	Definition/Sub-categories	Examples
Quality Assessment	Explicit statement judging the correctness or indicating an error, problem, or missing aspect in the submission under review.*	"correct", "not completely correct", "could not find any error", "Assignment says xy but you do yz", "xy is missing", "there are problems/gaps"
	Explicit general quality assessment statement that encompasses but is not limited to correctness. Not only pointing out errors.	"good solution", "everything is fine", "well done", "very nice", "perfect"
Style assessment*	Assessment of coding style, clarity, understandability, or readability of code.	"well indented", "elegant", "very tidy written", "nice, uncomplicated solution", "solution is understandable"
Praise*	Positive valuation/feedback	"you are more intelligent than me", "very good", "well done", "perfect", "I'm proud of you", "wonderful"
Coding tip*	(General) statements for improvements that do not affect correctness (not formatting).	"your solution is a bit complicated", "sorting functions can be done nicer", "no need for so many variable declarations"
Specific coding tip*	Specific coding tip containing keywords, variable names, or line references.	"'this' is not necessary here", "use i+=2 instead of i=i+2", "return in void method not needed (14 and 17)"
Correction*	(General) correction hint or pointing out an error	"All numbers are counted up, not only the odd ones."
Specific correction*	Specific correction containing keywords, variable names, or line references.	"replace i++ with i+=2", "Class Exam should use Generics", "After the bracket '}' in line 17, another bracket '}' must follow"
Formatting hint	Suggestion to improve code formatting or variable naming	"indent/format your code", "use camelcase", "put i++ in a new line"
Uncertainty	Explicit expressions of uncertainty.	"I am not sure", "Cannot say whether"
	Implicit expressions of uncertainty.	"In my opinion", "seems correct", "looks good", "I think that"
Learning	Expression of having learned or understood something	"... has become much clearer to me", "I haven't thought of ..."
Highlighting	Aspects explicitly highlighted on the submission	"short notation +=", "nice handling of invalid input", "elegant lamba [sic!]"
Empathy	Expressions of empathy for the submitter	"it was also hard for me at the beginning", "I feel your pain"
Negativity	Negatively formulated statement	"unfortunately not very well written", "completely cluttered" code
Emoji	Unicode or ASCII emoji	":-)", ";)", ":D", "^_^", "👍"
Thanks	Statement of thanks	"Thanks", "Mercii"
Questions*	Asking a question	"Doesn't line 18 have to be 'else if'?", "What is ...", "Why do you..."
Invalid term	Usage of an invalid technical term	"if-loop" (only)
Testing	Statement on actually testing the submission	"cannot compile", "when I execute your program", "works in Eclipse"
Motivation	Motivational statement	"keep it up"
Contact	Providing direct contact information	"contact me: [email-address]"
References	Reference to further material or task specification	"See lecture EiP-03 slides 24ff", "the task specification says ..."
Solution*	Complete (model) solution is provided	
Empty*	No sensible text was entered	"Bruh moment", ".", "-"

Table 4. Characterization of the reviews I (R: reviews, E: empty, QA: explicit correctness assessment of the code/explicit general quality assessment statement encompassing correctness, SA: style assessment, EM: Emoji, P: Praise, U: Uncertainty explicitly/implicitly)

Task	#R	#E	#QA (%)	#SA (%)	#EM (%)	#P (%)	#U (%)
3	43	1	23 (53)/13	11 (26)	10 (23)	20 (47)	1/3 (10)
4	39	1	14 (36)/20	9 (23)	11 (28)	24 (62)	1/5 (13)
5	35	1	14 (40)/13	9 (26)	9 (26)	24 (69)	2/4 (11)
6	30	0	19 (63)/9	6 (20)	6 (20)	17 (57)	2/4 (13)
7	22	1	9 (41)/7	3 (14)	6 (27)	10 (45)	2/3 (14)
8	17	0	8 (47)/7	5 (29)	6 (35)	10 (59)	0/2 (12)
9	16	0	7 (44)/6	4 (25)	2 (13)	8 (50)	2/4 (25)
10	13	1	5 (38)/5	1 (8)	0 (0)	10 (77)	0/2 (15)
Overall	215	5	99 (46)/80	48 (22)	50 (23)	123 (57)	10/27 (13)

Table 5. Characterization of the reviews II (T: Coding tip, F: Formatting hint, TS: Specific coding tip, C: Correction, CS: Specific correction, B: Both, Coding tip & Correction, CC: Suggested correction/coding tip fully correct (%: relative to # suggestions))

Task	#T (%)	#F (%)	#TS (%)	#C (%)	#CS (%)	#B	#CC (%)
3	11 (26)	6 (7)	9 (21)	9 (21)	6 (14)	0	17 (85)
4	5 (13)	1 (3)	5 (13)	7 (18)	4 (10)	1	9 (82)
5	10 (29)	2 (6)	7 (20)	5 (14)	3 (9)	3	12 (100)
6	9 (30)	0 (0)	8 (27)	14 (47)	11 (37)	6	13 (76)
7	4 (18)	0 (0)	3 (14)	4 (18)	2 (9)	0	7 (88)
8	1 (6)	0 (0)	1 (6)	6 (35)	4 (24)	1	5 (83)
9	3 (19)	0 (0)	1 (6)	5 (31)	4 (25)	1	6 (86)
10	2 (15)	0 (0)	2 (15)	2 (15)	2 (15)	1	3 (100)
Overall	45 (21)	9	36 (17)	52 (24)	36 (17)	1	72 (86)

"the big problem of your implementation is. . . " could only be found in 4 reviews whereas only one refers to the correctness and the other two to coding style — these reviews also contain constructive feedback or praise.

Explicit statements of uncertainty were found in 10 reviews. Noteworthy are 3 reviews in which the reviewers state that they could not solve the assignment either (twice for assignment 7, and once for assignment 9). Indications of implicit uncertainty are more prevalent, appearing in 27 reviews. These are mostly relativizations (such as "In my opinion" or "seems to work") and indicate that students did not check the correctness intensively. There are also cases where students are unsure about corrections such as "replace i++ with i+=2, then everything should be correct". The frequency is always around 12% with an outlier of 25% for assignment 9 (singly linked list). Almost all uncertainty statements are related to the assessment of the correctness and only 5 refer to suggestions.

General coding tips (not related to code formatting) were identified in 45 reviews and right from the beginning. Hints for code formatting were only given for first three coding assignments and significantly less than other coding tips. Concrete coding tips with code or referring to the submission, were given between 6 and 27% of the reviews. Corrections were given and errors pointed out in 52 reviews (24%, knowledge of mistakes, cf. Table 6) and occur at least as frequently as coding tips (except for assignments 3 and 5). Concrete steps for correction or examples for parts of the solution (subset of *Specific correction*) can be found in 24 reviews (11%, knowledge of how to proceed, cf. Table 6). Most coding tips and corrections (relatively) were given for the first object-oriented programming assignment (task no. 6). No clear trends over time are visible. Only 14 reviews contain both a correction and a coding tip. In 15 reviews, there are references to the assignment specification (7%, knowledge of task

constraints, cf. Table 6), and in 14 reviews there are references to the lecture material, or more information about the problem or a concept (6%, knowledge of concepts, cf. Table 6). Complete solutions of the reviewers were found in 6 reviews across all tasks (3%, knowledge of correct solution, cf. Table 6). The suggested coding tips and corrections are discussed in Sections 5.3 and 5.4.

Table 6. Characterization of the reviews regarding Narciss (2008) feedback categories: KR: knowledge of result/response; KM: knowledge of mistakes, KH: knowledge of how to proceed, KTC: knowledge of task constraints, KC: knowledge of concepts, KCR: knowledge of correct solution

	KR	KM	KH	KTC	KC	KCR
# Reviews (%)	99 (46)	52 (24)	24 (11)	16 (7)	14 (6)	6 (4)

There are only 5 reviews that can be identified as "empty". Three reviews contain only a dash or a dot, one contains only the emoji ":)", and one contains only the comment "bruh moment". Interesting is, how the reviewers reacted on the 12 submissions without intensive solution attempts: There is a submission with only the comment "I didn't save my work...", here the reviewer reacted with "Happens.Better Luck next time!" (not tagged as "empty"). The comment "bruh moment" was given for a submission with no visible solution attempt (task 10). For 4 almost empty submissions, empathy was shown such as "it was also hard for me at the beginning", and in 8 cases parts of the solution and/or hints for how to proceed were provided in the feedback.

Questions could only be found in seven reviews such as "Your program only outputs a 'Boom!'. Did you test the program before submitting it?", or "Doesn't line 18 have to be 'else if'?". Notable is one review in which the reviewer disclosed him or herself by providing contact information for further assistance ("If you have any further questions, please do not hesitate to contact me:") additionally to an extensive review with lots of detailed explanations.

Finally, in 14 reviews specific aspects of the solution were explicitly highlighted such as usage of "+=", "nice" handling of errors, definition of separate methods, or a comparator implemented in a "nice" way using a lambda expression. Furthermore, in 8 reviews students expressed that doing the review helped them, they have learned or understood something, or have not thought of a specific aspect (e.g. "I had not fully understood the principle of comparators before, but it has become much clearer to me with your solution." or "I particularly like the short notation +=, I'll remember it straight away"). Four of these comments were found on task 5 (counting specific chars in an array).

5.3. Review Correctness, Errors, and Corrections

Overall, the accuracy of the reviews (based on the correctness=4 and completeness=4 ratings) is 57%. The true positive rate (sensitivity) is 42% and the true negative rate (specificity) is 85%.

Only in 8 cases (across nearly all tasks, 15% of all corrections) incorrect corrections could be identified. The most common incorrect corrections that occur at least twice are: The assignment specification was incorrectly quoted or violated twice, twice the suggested correction would not change the correctness, and twice the output was not correctly checked. No errors were introduced by any coding tip.

Apart from the encoding issues, there are 31 syntactically incorrect submissions. For assignment 3, the class name did not match the filename twice — these errors

were not spotted by reviewers despite giving an coding tip or stating to have learned something. The main issue for assignment 7 (5 times) and 8 (3 times) were missing classes — these were just mentioned in three reviews explicitly and in one implicitly by pasting compiler errors without further explanation. Interestingly, one student said s/he cannot fully review due to a missing class and gave full correctness and completeness scores. There are a few other errors such as 5 times missing/superfluous closing brackets which were detected once explicitly and 3 times by acknowledging an incomplete submission, defining private variables or a constructor in the `main` method (both mentioned implicitly by recommending to looking into theory again, e.g. "Furthermore, I would highly recommend that you take another look at how to create objects."), and towards the 3 last assignments several partly empty submissions (majority acknowledged as only being solution attempts, e.g. "The solution is not complete, but I'm sure you know that."). Overall, in 15 of the 31 cases with real syntax errors these were not detected, 9 errors were explicitly mentioned, and in 7 cases some or other errors were mentioned or a solution provided.

Table 7. Classification and Detection of Errors (seen/frequency; Algo: error in algorithm, OF: output format error, WS: wrong signature/classname, I: incomplete solution)

No.	Algo.	#OF	#WS	#I
3	8/10	0/7	0/2	0/0
4	4/25	0/3	0/1	0/0
5	3/6	1/7	0/4	3/3
6	5/13	1/5	2/8	5/8
7	1/5	0/0	0/0	5/8
8	4/5	2/11	0/2	4/6
9	1/8	0/0	1/8	3/4
10	2/9	0/1	0/4	2/2

Errors violating the functional correctness and adherence to the specification are much more diverse, hence we clustered these (cf. Table 7). The numbers for *frequency* and *seen* in Table 7 mean, for example, that for assignment 3 there are ten submissions with at least one algorithmic error, while in eight reviews at least one of these was mentioned. The wrong signature/classname category does not consider wrong package names. Overall, students seem to be better in detecting algorithmic errors and incomplete submissions compared to output errors and errors in signatures/class names. Output (format) errors such as additional output or typos (reported/cases: e.g. for the third assignment: "Boom!" was required, but "BOOM!" delivered 0/6, and all numbers were printed on the same line instead of separate ones 0/1), or superfluous separator commas such as "1,2,3," instead of "1,2,3" are reported less frequently and errors in method signatures and class names only in very few cases. Noteworthy errors (counted as wrong algorithm) are for assignment 4 that the first number that was used as an input for the Collatz algorithm is often missing in the output (despite there was an example output in the assignment specification; these account for 3/22 of the overall 4/25 errors), for assignment 6 integer division was used instead of double division (3/7), for assignment 9 the sorting order was inversed (0/8; for one of these submissions another error was corrected), and for the last assignment the ISBN10/13 checks were wrong/incomplete (2/6 of the overall 2/9) or students caught their thrown exception in the very same method (0/2 additional to the 2/6). Only 11 reviews contain a statement that the students actually (tried) to test and execute the submission. In four reviews,

an invalid wording "if-loop" is used. Only in few cases all errors in the submission are mentioned in the reviews. No clear trends are visible.

5.4. Coding Style and Suggested Coding Tips

Most suggested coding tips can be counted to improving the readability or code quality. There are 13 specific coding style hints (7 of these concern code comments or indentation). The coding tips at least mentioned twice are: 4 times eliminating double checks of a condition in two following if-statements, 4 times reduction of variable use (e.g. defining a variable only to be used in an if-statement's condition), 3 times put code into a separate method, 3 times redundant code in if and else part, twice using the short version `++i`, twice reuse code (e.g. call an implemented/existing method instead of a reimplemention), twice variable naming, twice not every line needs a comment, and twice performance related (avoiding unnecessary type conversions). Interestingly, `if (condition) {} else {something}` was used twice and only mentioned once — same for nested if conditions instead of one boolean expression with an `&&` (*and*). Additional debug statements or having a `main` method in all classes were criticized once each. In two reviews students requested comments in future submissions.

6. Discussion

The reviews were qualitatively analyzed to get an overview of their characteristics (RQ1): Overall, the reviews are rather short as often reported in related work (Hamer et al., 2015, 2009; Heller & Bry, 2019; Politz et al., 2016; Strickroth, 2023) and the majority contains just an (explicit correctness or general) quality assessment statement which was also found by Politz et al. (2016) in their *LGTM* (Looks good to me) category. There is a high overlap of the categories *general quality assessment* and *praise* as there are many very short peer reviews such as "very good", "perfect", but the *praise* category also contains other positive valuation statements such as "I'm proud of you" and may also be used when there is an *explicit correctness* combined with a positive valuation statement. Judgments about coding style are less common. First year students seem to be more concerned with correctness, and experience with readable code may not yet be pronounced (cf. Hundhausen et al., 2009). This also manifests in shorter reviews for correct submissions than for incorrect ones. Elaborated feedback (such as knowledge of task constraints, concepts, and how to proceed, cf. Narciss, 2008) can be found, however, the majority is only informative (46% knowledge of response/result). Therefore, students seem to require additional guidance when writing reviews or incentives (cf. Indriasari et al., 2022). Only in very few cases, all the errors contained in a submission are discussed in the review. This is understandable, especially in the case of incomplete submissions where too many aspects would have to be considered, but also it might be advisable to focus on the most important aspects as teaching assistants would do (cf. Jeuring et al., 2022). Overall, the sentiment of the reviews seems to be neutral to positive, with positive emojis and praise in more than half of the reviews. This is in line with related work (cf. Cho et al., 2008, 2006; Hamer et al., 2015). The frequency of motivational statements is only 7% but seems to be comparable to the study of Indriasari et al. (2022). Overall, students are supportive and respectful as there are negligible many reviews with a negative tone. Submissions with no intensive solution attempt and empty reviews are not a significant issue, but this may be due to the specific scenario where participation was fully voluntary. Still,

students had to submit something as a review to stay in the peer review process to get feedback due to the strict exclusion rule. Against the hope of the authors, feedback addressing readability, comments, or indentation was only found rarely despite quite few code comments in the submissions. Questions, a key element in professional reviews (Bacchelli & Bird, 2013; Davila & Nunes, 2021), were seen only rarely, probably because dialogue was not possible. No off-topic aspects were found in the comments compared to one case in the study of Indriasari et al. (2022).

For most characteristics, no trends over the semester can be identified (RQ2). Coding tips and corrections can be found right from the beginning, and also the correctness of the corrections stays at a high level. The number of commented submissions increases over time. It was expected that also feedback would become more elaborated over the semester, because students receive higher and lower quality feedback themselves, learn from that and provide better feedback. However, this did not happen. A decreasing number of submissions is "normal" (in Germany, cf. Strickroth, 2023). Towards the end of the semester, the assignment complexity increased and required students to master not only the latest taught concepts but also the basic ones. This could be the reason why some characteristics such as concrete coding tips are so low for assignment 8 and 9, and uncertainty peaks for assignment 8.

Students seem to be better in detecting an incorrect submission as incorrect (specificity: 85%) compared to identify a correct submission as correct (sensitivity: 42%; RQ3). However, students often failed to spot more subtle errors related to output formatting and incorrect signatures/classnames. Also, "minor" errors in algorithms (cf. assignment 4 or 7) are only rarely detected. Nevertheless, students are quite good in detecting and providing support for incomplete submissions (aka. solutions attempts). This may indicate that students focus on the general solution strategy and do not look for all details. It seems as if students do not actually test the submissions but just read/skim them. A reason may be that there was no download link provided, however, most tasks required only a single file and the contents could be copied and pasted into an IDE easily. On the one hand, this can be problematic as subtle errors can render algorithms invalid (according to the specification or may be one of the key aspects of an assignment). But such kind of errors can easily be detected using automated tests. On the other hand, students can also provide feedback on algorithms for syntactically invalid or non-automatic testable submissions (due to errors in class names, packages, and signatures). Many reviews point out or provide useful hints for solving quality issues. Hence, peer reviews may be a good way to complement automatic feedback. But some misconceptions (such as catching thrown exceptions) were not detected or combining/inverting boolean expressions causing a bad coding style with nested if-conditions were rarely detected (cf. Oliveira, Keuning, & Jeuring, 2023). Overall, there are only 15% of incorrect corrections and no incorrect coding tips but 17% of the reviews contain indicators of uncertainty. Uncertainty was not mentioned in related research so far. Further research should investigate how uncertainty can be reduced and how students can be supported in writing their feedback (e.g. by providing them automatic test results or checklists for subtle aspects).

It should be noted that correctness and quality of the reviews are not the only goal of peer review: Often the primary goal is the value that arises from reviewing and writing a review (e.g. Hamer et al., 2009). Even if some coding tips are subjective or incorrect in some cases, it is still a good exercise to explicitly reflect on them and think about advantages and disadvantages (Dietz, Manner, Harrer, & Lenhard, 2018). Hence, the quality of given coding tips was not investigated and rated, but only analyzed how often specific feedback was given. Nevertheless, the quality of the review could be an

indication of the quality of the student's reflection (cf. Hamer et al., 2015), here further research is necessary as well as on how students react on (partly) incorrect feedback.

7. Threats to Validity

The inherent limitations of the qualitative research paradigm and thematic analysis apply. The very high agreement of 1 or not being able to calculate Cohen's κ value for several categories relates to the sparsity of these categories in the reviews (e.g. *thanks* only being found in 2 reviews) or the very clear definitions (e.g. *emoji* and *thanks* that could also be identified mechanically). Even for those categories where no κ value could be calculated, there was the perfect agreement that these categories do not apply to the samples in the evaluation dataset. It was decided to keep these "small" categories with low frequencies as these outline very interesting details. To mitigate possible ambiguities, all categories and definitions were intensively discussed as described in Section 4. Particularly for categories such as *quality assessment*, *praise* and *coding tip* which may be more prone to ambiguity, the agreement is at least substantial (Landis & Koch, 1977; McHugh, 2012). Hence, it could be argued that the inter-rater agreement is very good overall. Still, there could be isolated classification errors or biases of the coder even if everything was double checked.

The error categories presented in Section 5.3 and possible code improvements in Section 5.4 were developed by a single experienced researcher/programmer and may be subjective. Also, the classification in Table 7 may be too coarse grained; it was tried to mitigate this by providing details in the text. They can still be considered exploratory and provide a basis for comparisons in future research.

About 25% of the reviews were delivered by students after the exercise session took place where they were registered for. However, it is not known whether they attended the exercise session or another one. The authors do not think this introduces a significant bias, because in such a setting where peer reviews are homework assignments it is also unknown whether students have written their reviews on their own, jointly worked on their solution, or talked to other students about their solution or the reviews.

It could be argued that there is a selection bias because both the homework submission and the peer review were voluntary and only the more engaged students participated. For this research it is argued that this argument does not imply any bias, because it is seen as an authentic sample of this specific scenario. However, this aspect must be taken into account when comparing the results with mandatory or graded peer reviews (e.g. regarding "empty" reviews).

Finally, it may be argued that a study with untrained students and without external feedback of teaching assistants on the reviews is a limitation. However, such scenarios do exist and it also is up for debate what kind of (extensive) training is possible for first-semester students who are just learning to program. Even though researchers and practitioners expect such feedback to be of low quality, this research provides empiric evidence and a characterization of authentic first-semester student feedback that can be used for future comparisons, e.g. when different interventions are used or novel support features are provided. Also, the amount and type of training was often not clearly stated in related work and should be done more explicit in the future.

8. Conclusions and Outlook

In this paper, voluntary submissions and given peer feedback of a first-semester programming course are qualitatively analyzed. Overall, the reviews are neutral to positive and contain mostly information about the correctness but also corrections and useful coding tips. A lot of praise could be identified and only few submissions without intensive solutions attempts or "empty" reviews. Students seem to be better in checking algorithm "ideas" compared to details such as output formatting, and typos in class and method names. Most corrections are correct and coding tips did not introduce new errors. At the same time, uncertainty was found in the peer reviews quite often. Overall, no significant changes in the reviews have been identified over time — notable are, however, peaks in frequencies for quality assessments and corrections for the fourth coding assignment (may to be task dependent) and a decrease for coding tips and formatting hints over the semester. Also, the frequency of commented submissions increased over time (except for the last assignment).

The developed coding book and these results contribute to the understanding of student peer code review. The contributions help to address the identified issues and to develop targeted support for students to improve the quality of the reviews.

Further research is needed on three levels: First, how helpful is giving vs. receiving (even incomplete) feedback (cf. Wu & Schunn, 2023)? Second, how to improve the feedback? Here, worked-examples, training, model-solutions, test-results, or (detailed) checklists/rubrics may be provided and need to be evaluated which are better suited (under which circumstances). Other approaches may include discussion between submitters and reviewers, encouraging questions, or discussing given reviews in class or exercise sessions. Third, it may be interesting to sample some students and analyze their peer review behavior over time instead of aggregated per assignment. Finally, future literature surveys and papers should clearly indicate the type/amount of training offered to students and the used prompts.

Acknowledgements

We thank all students who participated and allowed their data to be used for (this) research! We also thank the reviewers for their valuable feedback.

Funding

This work was supported by the German Federal Ministry of Education and Research (Bundesministerium für Bildung und Forschung, BMBF) under the grant number [16DHBKI013] (project AIM@LMU).

Disclosure statement

No potential conflict of interest was reported by the author(s).

Contribution Statement

Sven Strickroth conceptualized the analysis, conducted the coding, and took the lead in writing the draft and final version of the paper. Imen Azaiz conducted the coding for the inter-rater analysis, provided feedback on the categories, helped to come up with their final definitions, and contributed to the analysis of errors in the code submissions, the writing of the initial draft and proofreading. Both authors approved the final version of the manuscript.

References

Asri, I. E., Kerzazi, N., Uddin, G., Khomh, F., & Idrissi, M. J. (2019, oct). An empirical study of sentiments in code reviews. *Information and Software Technology*, *114*, 37–54. doi:10.1016/j.infsof.2019.06.005

Azaiz, I., Deckart, O., & Strickroth, S. (2023, dec). AI-Enhanced Auto-Correction of Programming Exercises: How Effective is GPT-3.5? *International Journal of Engineering Pedagogy (iJEP)*, *13*(8), 67–83. doi:10.3991/ijep.v13i8.45621

Azaiz, I., Kiesler, N., & Strickroth, S. (2024). Feedback-Generation for Programming Exercises With GPT-4. In *Proceedings of the 2024 Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2024), July 8–10, 2024, Milan, Italy.* ACM. doi:10.1145/3649217.3653594

Bacchelli, A., & Bird, C. (2013). Expectations, outcomes, and challenges of modern code review. In *35th International Conference on Software Engineering (ICSE)* (pp. 712–721). doi:10.1109/ICSE.2013.6606617

Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, *3*(2), 77-101. doi:10.1191/1478088706qp063oa

Brown, T., Narasareddygari, M. R., Singh, M., & Walia, G. (2019). Using Peer Code Review to Support Pedagogy in an Introductory Computer Programming Course. In *2019 IEEE Frontiers in Education Conference (FIE)* (pp. 1–7). doi:10.1109/FIE43999.2019.9028509

Cho, K., Chung, T. R., King, W. R., & Schunn, C. (2008, mar). Peer-based computer-supported knowledge refinement. *Communications of the ACM*, *51*(3), 83–88. doi:10.1145/1325555.1325571

Cho, K., & Schunn, C. D. (2007). Scaffolded writing and rewriting in the discipline: A web-based reciprocal peer review system. *Computers & Education*, *48*(3), 409–426.

Cho, K., Schunn, C. D., & Charney, D. (2006, jul). Commenting on writing. *Written Communication*, *23*(3), 260–294. doi:10.1177/0741088306289261

Choudhury, R. R., Yin, H., Moghadam, J., & Fox, A. (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

Cohen, J. (1960, apr). A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, *20*(1), 37–46. doi:10.1177/001316446002000104

Davila, N., & Nunes, I. (2021, jul). A systematic literature review and taxonomy of modern code review. *Journal of Systems and Software*, *177*, 110951. doi:10.1016/j.jss.2021.110951

de Raadt, M., Lai, D., & Watson, R. (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* (Vol. 88, p. 53–64). AUS: Australian Computer Society, Inc.

Dietz, L. W., Manner, J., Harrer, S., & Lenhard, J. (2018, Mar). Teaching clean code. In *Proceedings of the 1st Workshop on Innovative Software Engineering Education.*

Dolezal, D., Posekany, A., Roschger, C., Koppensteiner, G., Motschnig, R., & Pucher, R. (2018, feb). Person-centered learning using peer review method – an evaluation and a concept for

student-centered classrooms. *International Journal of Engineering Pedagogy (iJEP)*, *8*(1), 127–147. doi:10.3991/ijep.v8i1.8099

Hamer, J., Purchase, H., Luxton-Reilly, A., & Denny, P. (2015). A comparison of peer and tutor feedback. *Assessment & Evaluation in Higher Education*, *40*(1), 151-164. doi:10.1080/02602938.2014.893418

Hamer, J., Purchase, H. C., Denny, P., & Luxton-Reilly, A. (2009, aug). Quality of peer assessment in CS1. In *Proceedings of the fifth international workshop on computing education research workshop.* ACM. doi:10.1145/1584322.1584327

Hanrahan, S. J., & Isaacs, G. (2001, may). Assessing self- and peer-assessment: The students views. *Higher Education Research & Development*, *20*(1), 53–70. doi:10.1080/07294360123776

Hattie, J., & Timperley, H. (2007, mar). The power of feedback. *Review of Educational Research*, *77*(1), 81–112. doi:10.3102/003465430298487

Heller, N., & Bry, F. (2019). Organizing peer correction in tertiary stem education: An approach and its evaluation. *International Journal of Engineering Pedagogy (iJEP)*, *9*(4), 16–32.

Hundhausen, C., Agrawal, A., Fairbrother, D., & Trevisan, M. (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* (p. 291–295). doi:10.1145/1508865.1508972

Indriasari, T. D., Denny, P., Lottridge, D., & Luxton-Reilly, A. (2022, September). Gamification improves the quality of student peer code review. *Computer Science Education*, *33*(3), 458–482. doi:10.1080/08993408.2022.2124094

Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2020a, may). Gamification of student peer review in education: A systematic literature review. *Education and Information Technologies*, *25*(6), 5205–5234. doi:10.1007/s10639-020-10228-x

Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2020b, sep). A review of peer code review in higher education. *ACM Transactions on Computing Education*, *20*(3), 1–25. doi:10.1145/3403935

Indriasari, T. D., Luxton-Reilly, A., & Denny, P. (2021, jun). Investigating accuracy and perceived value of feedback in peer code review using gamification. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1.* ACM. doi:10.1145/3430665.3456338

Jeuring, J., Keuning, H., Marwan, S., Bouvier, D., Izu, C., Kiesler, N., ... Sarsa, S. (2022, dec). Towards giving timely formative feedback and hints to novice programmers. In *Proceedings of the 2022 working group reports on innovation and technology in computer science education.* ACM. doi:10.1145/3571785.3574124

Keuning, H., Jeuring, J., & Heeren, B. (2018, September). A systematic literature review of automated feedback generation for programming exercises. *ACM Transactions on Computing Education*, *19*(1). doi:10.1145/3231711

Kiesler, N., Lohr, D., & Keuning, H. (2023). Exploring the potential of large language models to generate formative programming feedback. In *2023 IEEE Frontiers in Education Conference (FIE)* (p. 1-5). doi:10.1109/FIE58773.2023.10343457

Landis, J. R., & Koch, G. G. (1977, mar). The measurement of observer agreement for categorical data. *Biometrics*, *33*(1), 159. doi:10.2307/2529310

Le, N.-T., Loll, F., & Pinkwart, N. (2013, jul). Operationalizing the continuum between well-defined and ill-defined problems for educational technology. *IEEE Transactions on Learning Technologies*, *6*(3), 258–270. doi:10.1109/tlt.2013.16

Liu, X., & Woo, G. (2020, feb). Applying code quality detection in online programming judge. In *Proceedings of the 2020 5th International Conference on Intelligent Information Technology.* ACM. doi:10.1145/3385209.3385226

Mayring, P. (2001). Combination and integration of qualitative and quantitative analysis. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research*, *Vol 2*. doi:10.17169/FQS-2.1.967

McHugh, M. (2012, 10). Interrater reliability: The kappa statistic. *Biochemia*

medica : časopis Hrvatskoga društva medicinskih biokemičara / HDMB, *22*, 276-82. doi:10.11613/BM.2012.031

Narciss, S. (2008). Feedback strategies for interactive learning tasks. In *Handbook of research on educational communications and technology* (3rd ed., pp. 125–144).

Nicol, D. (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

Nicol, D., Thomson, A., & Breslin, C. (2013, may). Rethinking feedback practices in higher education: a peer review perspective. *Assessment & Evaluation in Higher Education*, *39*(1), 102–122. doi:10.1080/02602938.2013.795518

Oliveira, E., Keuning, H., & Jeuring, J. (2023, jun). Student code refactoring misconceptions. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1.* ACM. doi:10.1145/3587102.3588840

Politz, J. G., Collard, J. M., Guha, A., Fisler, K., & Krishnamurthi, S. (2016, February). The sweep: Essential examples for in-flow peer review. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (pp. 243–248). ACM. doi:10.1145/2839509.2844626

Reily, K., Finnerty, P. L., & Terveen, L. (2009, may). Two peers are better than one. In *Proceedings of the 2009 ACM international conference on supporting group work.* ACM. doi:10.1145/1531674.1531692

Rybarczyk, R., & Acheson, L. (2019, feb). Interactive peer-led code reviews in CS2 curricula. In *Proceedings of the 50th ACM technical symposium on computer science education.* ACM. doi:10.1145/3287324.3287442

Smith, J., Tessler, J., Kramer, E., & Lin, C. (2012, sep). Using peer review to teach software testing. In *Proceedings of the ninth annual international conference on international computing education research.* ACM. doi:10.1145/2361276.2361295

Sripada, S., Reddy, Y. R., & Sureka, A. (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* (pp. 3–6). doi:10.1109/CSEET.2015.8

Stalhane, T., Kutay, C., Al-Kilidar, H., & Jeffery, R. (2004). Teaching the process of code review. In *2004 australian software engineering conference. proceedings.* (Vol. 5, pp. 271–278). IEEE. doi:10.1109/aswec.2004.1290480

Strickroth, S. (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)* (pp. 498–504). ACM. doi:10.1145/3587102.3588802

Strickroth, S., & Bry, F. (2022). The Future of Higher Education is Social and Personalized! Experience Report and Perspectives. In *Proceedings of the 14th International Conference on Computer Supported Education - Volume 1: CSEDU* (Vol. 1, pp. 389–396). SciTePress. doi:10.5220/0011087700003182

Strickroth, S., & Holzinger, F. (2022). Supporting the Semi-Automatic Feedback Provisioning on Programming Assignments. In *Methodologies and Intelligent Systems for Technology Enhanced Learning, 12th International Conference* (pp. 13–19). Cham: Springer International Publishing. doi:10.1007/978-3-031-20617-7_3

Strickroth, S., Olivier, H., & Pinkwart, N. (2011). Das GATE-System: Qualitätssteigerung durch Selbsttests für Studenten bei der Onlineabgabe von Übungsaufgaben? In H. Rohland, A. Kienle, & S. Friedrich (Eds.), *Tagungsband der 9. e-Learning Fachtagung Informatik (DeLFI)* (pp. 115–126). Bonn, Germany: Gesellschaft für Informatik e.V. Retrieved from `https://dl.gi.de/handle/20.500.12116/4740`

Strickroth, S., & Striewe, M. (2022, November). Building a Corpus of Task-based Grading and Feedback Systems for Learning and Teaching Programming. *International Journal of Engineering Pedagogy (iJEP)*, *12*(5), 26–41. doi:10.3991/ijep.v12i5.31283

Sun, Q., Wu, J., Rong, W., & Liu, W. (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

Søndergaard, H., & Mulder, R. A. (2012). Collaborative learning through formative peer review: pedagogy, programs and potential. *Computer Science Education*, *22*(4), 343-367. doi:10.1080/08993408.2012.728041

Topping, K. (1998, sep). Peer assessment between students in colleges and universities. *Review of Educational Research*, *68*(3), 249–276. doi:10.3102/00346543068003249

Turner, S. A., Pérez-Quiñones, M. A., & Edwards, S. H. (2018, sep). Peer review in CS2. *ACM Transactions on Computing Education*, *18*(3), 1–37. doi:10.1145/3152715

Wang, Y., Yijun, L., Collins, M., & Liu, P. (2008, mar). 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. doi:10.1145/1352135.1352171

Wu, Y., & Schunn, C. D. (2023, apr). Passive, active, and constructive engagement with peer feedback: A revised model of learning from peer feedback. *Contemporary Educational Psychology*, *73*, 102160. doi:10.1016/j.cedpsych.2023.102160