TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 06/11 pp423–434
DOI: 10.26599/TST.2018.9010109
Volume 24, Number 4, August 2019

Formative Assessment of Programming Language Learning Based on Peer Code Review: Implementation and Experience Report

Qing Sun, Ji Wu*, Wenge Rong, and Wenbo Liu

Abstract: In programming courses, the traditional assessment approach tends to evaluate student performance by scoring one or more project-level summative assignments. This approach no longer meets the requirements of a quality programming language education. Based on an upgraded peer code review model, we propose a formative assessment approach to assess the learning of computer programming languages, and develop an online assessment system (OOCourse) to implement this approach. Peer code review and inspection is an effective way to ensure the high quality of a program by systematically checking the source code. Though it is commonly applied in industrial and open-source software development, it is rarely taught and practiced in undergraduate-level programming courses. We conduct a case study using the formative assessment method in a sophomore level *Object-Oriented Design and Construction* course with more than 240 students. We use Moodle (an online learning system) and some relevant plugins to conduct peer code review. We also conduct data mining on the running data from the peer assessment activities. The case study shows that formative assessment based on peer code review gradually improved the programming ability of students in the undergraduate class.

Key words: peer code review; programming language learning; dynamic designation; formative assessment

1 Introduction

For a considerable quantity of courses in colleges and universities, comprehensive inspection of course assignments submitted by students is made difficult by an insufficient number of teachers relative to a large number of students. For the students, this gives rise to a utilitarian approach to completing the assignments, manifesting as lack of attention to the quality of learning. The students' homework cannot be fully assessed, feedback is often overdue, and there is little time for corrections. Consequently, there is only little possibility for the students to make improvements and for the course to achieve its desired educational effect. Peer assessment, an educational arrangement in which students judge each other's performance quantitatively and/or qualitatively[1], has been increasingly integrated in educational settings as a strategy to foster student learning. In peer assessment, students can be involved in assessing other students work or in assessing the contribution or performance of other students within the same group.

Peer assessment can be further classified into formative or summative assessment. Formative assessment aims at monitoring students' learning capabilities, providing them with ongoing feedback, and improving their learning experience in response[2]. Summative assessment, in contrast, evaluates students' capabilities at the end of an instructional unit[3].

Computer programming language learning is a typical example of a problem-oriented course. The assessment of outcomes from a programming language learning course is a challenging task. Traditional assessment methods are not adaptive to the evolving developments in computer programming

• Qing Sun, Ji Wu, Wenge Rong, and Wenbo Liu are with the School of Computer Science and Engineering, Beihang University, Beijing 100191, China. E-mail: sunqing@buaa.edu.cn; wuji@buaa.edu.cn; w.rong@buaa.edu.cn; liuwenbo@buaa.edu.cn.

∗ To whom correspondence should be addressed.

Manuscript received: 2018-05-19; revised: 2018-06-19; accepted: 2018-06-26

language education; new teaching approaches such as collaborative learning, Project-Based Learning (PBL), e-Learning, and m-Learning are endeavoring to explore new ways to boost learning outcomes[4, 5]. Peer assessment has been successfully used in a variety of computer programming courses for several years, mostly in the form of peer code review.

Code review is a quality assurance approach at the source code level. It originated from software engineering and has been a common programming practice for many years[6]. Software development communities benefit from the various light-weight and heavy-weight code review processes[4]. Fagan[7] first introduced the concept of code inspection and code review. Afterwards, the feasibility and efficiency of code review were studied and evaluated by many researchers. One related study[8] revealed that while finding bugs has been generally regarded as the main motivation to conduct reviews, they provide additional benefits such as knowledge transfer, increased team awareness, and the creation of alternative solutions to problems.

Some computer science and engineering educators have already benefited from peer review practices[9]. Li[10] demonstrated one way of incorporating a code review process into the assessment of programming language course, designing the code review process, which provided a positive environment and an opportunity to facilitate communication among students. Evaluation data was collected by personal observation, informal interviews, and a survey with rating scale questionnaires, while the evaluation was based on a Strengths, Weaknesses, Opportunities, and Threats (SWOT) analysis. Wang et al.[4] developed an online assessment system called EduPCR and used a novel approach to assess the learning of computer programming languages. Two questionnaire surveys and two interviews were conducted. The survey data and the interview report indicated that this assessment approach demonstrated a high level of practical value in assessing student learning outcomes in programming languages. Hundhausen et al.[11] developed an active learning approach for computing education called Pedagogical Code Review. The results provided evidence that Pedagogical Code Review could promote positive attitudinal shifts, and hone skills in critical reviewing, teamwork, and communication. Sripada et al.[12] conducted a study on the use of peer code review in an introductory Software Engineering course consisting of more than 200 sophomores and presented their experiences, findings, and challenges. Their experiments and survey revealed that employing peer code review in an undergraduate class had several learning benefits, such as improvement in coding skills, program comprehension abilities, knowledge of coding standards, and peer communication. Wang et al.[13] designed a multi-peer assessment platform for programming language learning, taking account of group non-consensus and personal radicalness. The survey results show significant improvements in students' learning outcomes. Positive feedback from using peer review, such as a better learning experience and more efficient learning outcomes, have been reported in some other works[14–17].

The effectiveness and benefits of peer assessment are very attractive. However, teachers do not adopt peer assessment until they believe that a peer assessment process is dependable and assessment results are reliable. One specific concern is that personal bias is an intrinsic and inevitable issue in peer assessment systems, affecting the reliability and fairness of peer assessment. In this research, we design and implement a formative peer assessment approach based on a Peer Code Review (PCR) model, in which some methods are provided for alleviating personal bias in a peer assessment setting. We also explore different approaches to enhance assessment reliability and improve students learning outcomes without adding too much of a burden on students and on the peer assessment process. The results are evaluated using the actual running data of the course.

2 PCR Model and Implementation System

The *Object-Oriented Design and Construction* course is a compulsory course for sophomore students majoring in Computer Science and Technology at Beihang University. The objective of this course is to develop skills in object-oriented programming. In each semester, the course contains more than 10 incremental object-oriented programming training tasks, all of which are assessed based on peer code review. An implementation system (OOCourse) is deployed in an online learning environment (Modular object-oriented dynamic learning environment (Moodle)) to support the formative peer assessment process.

2.1 PCR model

To facilitate the discussion of this formative assessment approach, we define a PCR model to capture the common concepts, data structure, roles, and activities of a typical peer assessment system. Though the model is defined primarily for peer assessment in a specific object-oriented course employing Java as the instructional programming language, it could be generalized and adapted to other programming language courses.

There are several roles, activities, and data structures in the model. Roles are defined as shown in Table 1.

Documents in the model are reclassified as follows:

(1) "Manuscript code" is program code written and submitted by an author; the manuscript code should be a program that has passed the steps of compiling, building, and testing by the author himself.

(2) "Assignment guideline" is written by the instructors. Released as soon as a new assignment is published, the guideline gives detailed explanations of the requirements stated in the assignment, and it helps the authors to reach consistent interpretations of the programming requirements.

(3) "Public test suite" is a set of test cases prepared by the instructors and teaching assistants and released together with the assignment guideline. The public test suite is used for common testing in the peer code review process and is not made visible until the assignments go into the corresponding stage. The tests help to detect whether a submitted manuscript is invalid; a valid manuscript code should be able to pass any test case in the public test suite, while an invalid manuscript should be removed from the peer code review process and not go further into competitive testing.

Table 1 Roles defined in the PCR model.

Author	A student who writes program and waits for code review activity by someone else
Reviewer	A student who reviews the code written by an author
Instructor	A teacher takes the responsibility for giving assignments and setting the responding schedules. The quality of experiment is supervised by instructors.
TA	Teaching Assistant (TA) is usually a graduate student or senior grade student who works as a teacher assistant, responsible for assignment grading and quality assurance

(4) "Argument on bugs" is a record of arguments on the bug report and is submitted by the author when he refuses to accept the review results given by the reviewer.

(5) "Technical blog" is written by the students at the end of each unit, to conclude the design and testing of the final three programming assignments. The technical difficulties met with in the assignments are to be summarized in the technical blogs.

At different stages of each assignment, students play the roles of both author and reviewer. Only a user with an instructor or a TA role can inspect and grade a student submission. The assessment process begins with a new assignment from a teacher and ends with score checking by students. Instructors take charge of publishing an assignment and setting the evaluation requirements together with the assignment guideline. A student acts as an author to submit his or her manuscript and as a reviewer to evaluate a manuscript code that they receive from another student. All reviewers are required to conduct both a common test (using the public test suite) and a competitive test. In the peer code review, reviewers play the role of reporting bugs in the received manuscript code. The received manuscript code will then be scored automatically according to the number and severity of the reported bugs. Each student can see the bugs reported by the student acting as their reviewer, and the bugs that they report acting as a reviewer. The final peer assessment results will be displayed at the end of each assignment. An activity graph for each assignment is roughly shown in Fig. 1.

Each assignment cycle is divided into five stages: (1) classroom teaching, (2) programming, (3) manuscript code submission, (4) double-blind peer review, and (5) score approval. In the first stage, the teacher will assign the task at the end of class. All of the assigned programming tasks and relevant resources are published on the teaching system. Students can access the resources to complete the programming tasks independently, and then submit the program code to the corresponding Workshop module before the deadline. At this point, students are allowed to enter the double-blind peer code review stage. Each student reviews the manuscript code assigned to them and submits the review results. To be more specific, after each manuscript has been appointed a reviewer, it will first be tested against the public test suite. Only if it passes at least one functional public test case is it regarded as

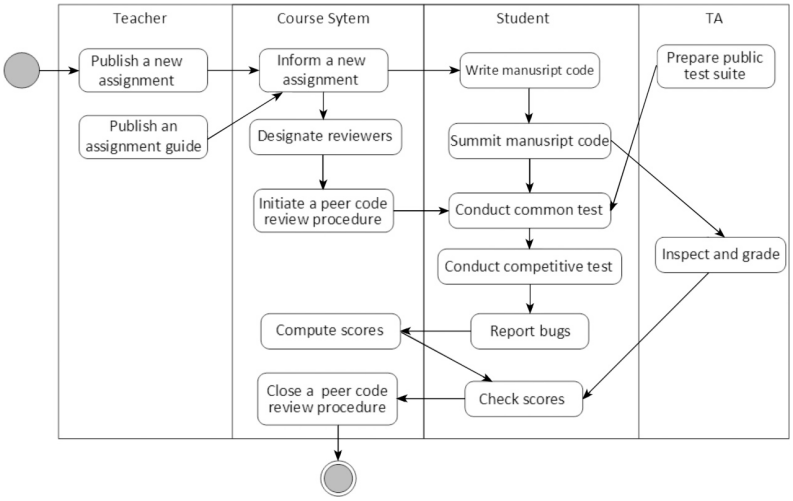


Fig. 1 Activities of different roles in the PCR model.

a valid manuscript and thus qualifies to continue the double-blind peer code review, followed by competitive testing on the test cases fully designed by the reviewer.

At the end, the results of the peer code review stage are collected to serve as input to the score approval stage. A student score consists of the programming writing score and testing score. The former is calculated by the bugs that he received as an author, while the latter is calculated by the bugs that he reported as a reviewer. The calculation of scores is completed by the system automatically, with the score for each student consequently given as feedback. At the end of the unit, the students are required to write technical blogs to summarize the completed tasks. The teaching system records data on all student activities in all of the above stages except the classroom session. These data are stored in the form of logs in the OOCourse system.

2.2 Model system

We describe an implementation of this model used in our programing language learning experiments. The formative peer assessment is implemented on the OOCourse system, which is developed based on Moodle, a famous online learning system. Developed in Australia by Martin Dougiamas, Moodle is based on constructivist education theories and has become a widely-used free and open source Course Management System[18]. In Moodle, the Workshop module is a powerful feature for supporting peer review. Our OOCourse makes use of the Workshop module, the discussion area module, the blog module, the course resource module, and the argument zone module, in which the argument zones match one-to-one to the workshops. Assignment submission, double-blind peer code evaluation, and the score approval phase are set-up in the workshop module according to the assignment process. Throughout an assignment cycle, students will use the course resource module to download courseware and assignment guide resources during the self-study stage. Meanwhile, they are allowed to carry on exchanges relating to relevant course content in the discussion module. During the assignment submission phase, the students will upload their manuscript code to the OOCourse website using the upload component in the workshop module. In the double-blind peer code review phase, each student will review another students manuscript code (as assigned by the system) and submit a testing report. If a student has any objections to bugs reported in their manuscript, they can appeal in the argument zone module during the score approval phase. At the end of each unit (three-assignment cycle), students are required to post their work experience in the blog module (as shown in Fig. 2).

3 Quality Assurance of the Peer Assessment

During the design and implementation of the formative peer assessment approach, some interesting research phenomena arise, including the particular challenges involved in assessment of programming language education as well as issues in student motivation and learning behavior. Therefore, we investigate some critical issues related to non-consensus, radicalness, and plagiarism. We intend to focus on quality assurance in the assessment process and the incentive mechanism. To minimize the negative aspects of a formative peer assessment approach, we implemented the following

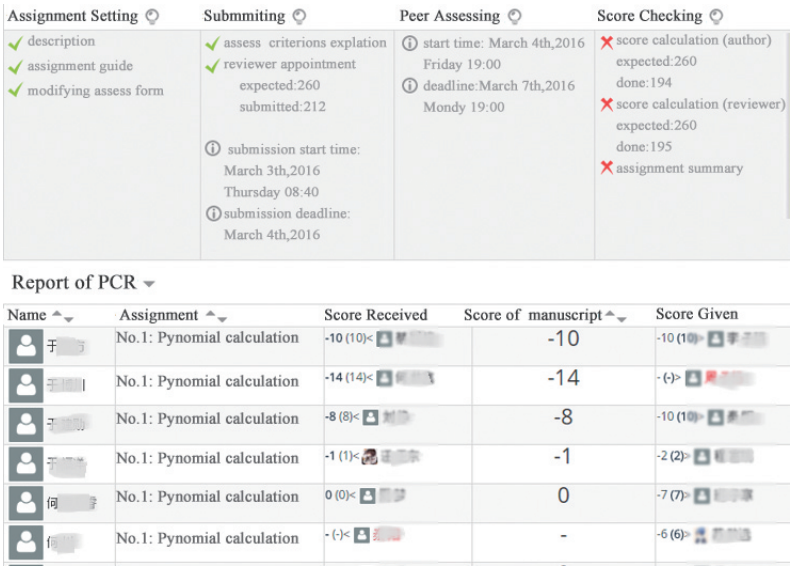


Fig. 2 Peer code review model supported by course system.

three functions in the OOCourse system: (1) a system of arbitration for non-consensus on reported bugs, (2) an anti-plagiarism mechanism, and (3) a dynamic reviewer appointment strategy.

3.1 Arbitration for non-consensus on reported bugs

Reporting bugs is not the ultimate goal of formative peer assessment. We are committed to making students aware of the bugs reported and to learn lessons from them. However, non-consensus is a common issue in peer assessment, and it has many causes. Students might apply different rules in deciding whether two bugs are the same or different. Furthermore, different students have different starting points and knowledge backgrounds. Students tend to hold different or even opposite opinions towards reported bugs, with varying views on issues such as coding style, perspectives for interpretation, variable naming conventions, environment configurations, program execution scenarios, the structure of program input, etc. The solving of non-consensus is important to the reliability of peer assessment. Unresolved disagreements will give rise to arguments between reviewers and authors, and could increase distrust among students to levels that would ultimately harm the successful running of a course. This might seriously influence the students' enthusiasm for, and attitude towards, the learning process, since the results from peer-review will contribute to their scores.

Therefore, we further extend the PCR model by adding a sub-flow for the arbitration of non-consensus. We provide an appeal module in OOCourse system to limit the problem. The arbitration follows the double-blind approach of peer code review. If an author does not agree with a reported bug, they can initiate an appeal procedure by raising an argument post in the system to the respective reviewer anonymously, providing reasonable arguments for their disagreement. The author and reviewer can discuss the arguments and provide relevant evidence for their point of view, with the arguments only visible to both parties, TAs and teachers. There are three possible outcomes: (1) the author and reviewer agree that the reported bug is invalid; in this case the bug needs to be removed from the list, and the corresponding scores need to be updated; (2) the author and reviewer agree that the reported bug is valid, and no change is made; (3) no agreement is attained; in this cases a TA or an instructor can intervene and arbitrate.

3.2 Anti-plagiarism mechanism

Since each student is set the same assignment, plagiarism has the potential to illegitimately increase scores. We apply an effective form of plagiarism detection in the OOCourse system to ensure the credibility of the assessment results. When an author submits manuscript code in the OOCourse system, a plagiarism detection result will be returned within one hour. If the similarity of the submitted code to some other program code in the data base is higher than the threshold setting, the submission will be

treated as a possible case of plagiarism. The author then has to respond to the claim of plagiarism. Only those submissions that pass plagiarism detection will be regarded as valid submissions and therefore be allowed to go into the double-blind peer code review. Figure 3 explains the extensions we made to the PCR model, integrating non-consensus arbitration and the anti-plagiarism mechanism.

3.3 Dynamic reviewer appointment strategy

At the beginning of the peer code review process, each manuscript should be appointed a reviewer. After ranking students into different categories (high, medium, low, etc.) according to their programming performance, there are two common strategies for designating reviewers. A stepwise designation strategy means that higher-ranked students review programs written by lower-ranked students, while a teacher reviews programs written by the highest-ranked students. Comparatively, an equal-rank designation strategy assigns students of equal rank to review each other's work, which has the benefit of the maximum exercise of each student's ability. We design a dynamic designation algorithm based on students' historical performance distribution before each assignment. In contrast to the method of distributing the manuscripts based only on historical homework scores, our dynamic reviewer appointing algorithm will make a reasonable allocation in which the programming ability and historical behavior of the author and reviewer are similar.

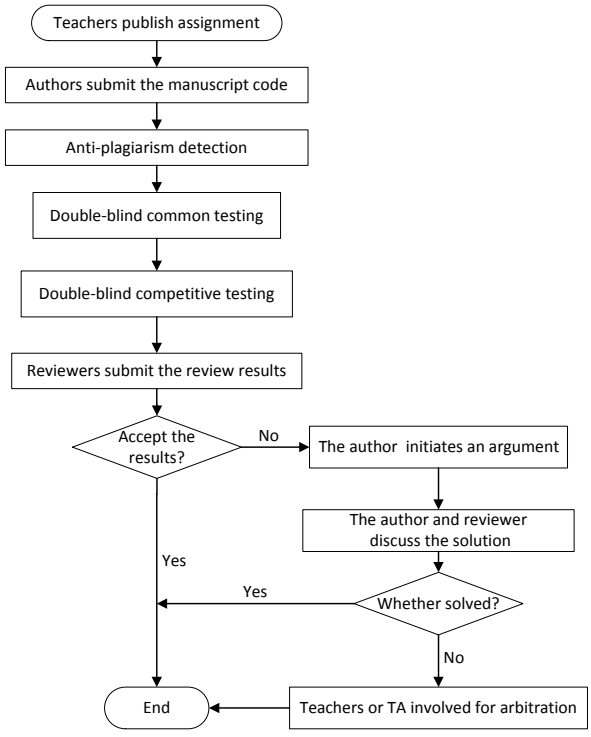


Fig. 3 Flow chart of extended PCR model.

In this paper, the reviewer appointing experiment is carried out on the submitted manuscripts. We extract certain features of both the peer assessment results and student behavior from historical data. The features cover the peer code review scores, the pass rate of public test cases, the number of bugs reported, the frequency of manuscript code submission, the activity of students in the discussion module, and the number of technical blogs published. The above features are summarized in Table 2.

The programming ability of the student is reflected in y_1 and y_2. Feature y_3 indicates the testing ability. The behaviors of the student are represented by y_4, y_5, and y_6.

In our reviewer appointing experiment, we intended to firstly group students according to their programming ability, testing ability, and the similarities in behavior from the historical data collected by the OOCourse system. This kind of data driven approach can help to find hidden groups from unlabeled data. We made reference to K-Means clustering, a typical kind of unsupervised learning method for grouping, when designing the appointment algorithm. We divide the students into five groups, so that the value of k in the algorithm is set to 5. Similar to the K-Means algorithm, we group the students according to the calculation of the Euclidean distance between each of them and the pre-selected mass center of the sample. The Euclidean distance formula between two students is shown in Eq. (1):

$$D = \sqrt{\sum_{i=1}^{N}(X_{y_i} - Y_{y_i})^2} \tag{1}$$

where D is the Euclidean distance between two samples, X and Y stand for the two student samples. y_i is the i-th feature of the sample, N is the space

Table 2 Features evaluating the programming ability and learning behavior.

Feature	Explanation
y_1	Peer code review scores received by the author
y_2	Passing rate of public use cases
y_3	Number of bugs reported
y_4	Frequency of manuscript code submission
y_5	Number of publish and reply in the discussion area
y_6	Number of valid blogs published

dimension of sample features.

For each student, we obtain the Euclidean distance from the five pre-selected mass centers. We divide the students into different groups with the principle of nearest classification sampling. If a sample has the same distance to more than one mass center, we assign it into one of the proximate groups randomly. Then we repeatedly calculate the mass centers in each group, with Eq. (1) used again to calculate the Euclidean distance between the group sample and its mass center. In this manner, the divided groups are updated. This procedure is repeated until the sample distribution remained stable.

The student grouping algorithm for the appointment of reviewers is shown in Algorithm 1. After grouping, students with matching programming and testing abilities are clustered into the same group. Additionally, our appointing algorithm based on students' behavior features guarantees that students in the same group have similar learning behaviors and attitudes. Within each group, reviewers are appointed randomly. We use the method of annotation to exclude the case of students A and B acting mutually as author and reviewer. For each assignment, by collecting the required data and extracting features accordingly, the appointing algorithm is adaptive as the course continues.

4 Findings of the Learning Outcomes

The formative assessment approach has been implemented in the *Object-Oriented Design and Construction* course since 2015. In each semester, more than 240 sophomore students were involved in the course. Throughout the semester, students were required to finish 12 programming assignments and 4 summary blogs (Nos. 4, 8, 12, and 16) in order of ascending difficulty. The topics for all of the programming assignments are listed in Table 3.

Algorithm 1 Reviewer appointing oriented student grouping algorithm

Require:
The number of groups that plan to assign, K
The students sample set, S
Ensure:
The divided groups of the sample, $S_1, S_2, \ldots, S_K$;
1: Divide sample S equally into K subsets
2: Mass centers set M is selected by nearest principle, $M = M_1, M_2, \cdots, M_K$;
3: Calculate the Euclidean distance between sample S_i and the K mass centers, $D_{ij} = (S_i, M_j)$;
4: if $D_{ij} = \min\{D\}$, put S_i into group K_j;
5: else if $D_{ij} = D_{i\acute{j}}$, randomly put S_i into groups K_j or $K_{\acute{j}}$;
6: For each sample S_i in S, repeat Steps 3–5;
7: Repeat Steps 2–6 until the sample distribution no longer changes.
8: **return** S;

4.1 General statistics

We use the collected data of the *Object-Oriented Design and Construction* course for the 2017–2018 academic year to evaluate the learning outcomes. The general results are shown in Fig. 4.

In this study, the students' programming ability refers to the program construction ability, which is mainly reflected in their performance. As shown in Fig. 4, the total number of students involved in the peer assessment is 243. Among all 10 programming tasks, none of them achieved 100% completion. Suppose the number of submissions is n, and the number of instances of plagiarism is m. In this paper, we define the number of valid manuscript code (p) as the difference between

Table 3 12 assignments in programming throughout the semester.

No.	Programming assignment topic
1	Pynomial calculation
2	Single threaded elevator
3	ALS (A Little Smart) single threaded elevator
5	Multithreaded elevator
6	File monitoring
7	The taxi
9	Taxi and JSF
10	Taxi at the intersection
11	Traceable taxi
13	ALS single threaded elevator specifications and coverage test
14	Proof of correctness of single elevator based on specification
15	Specify design in UML

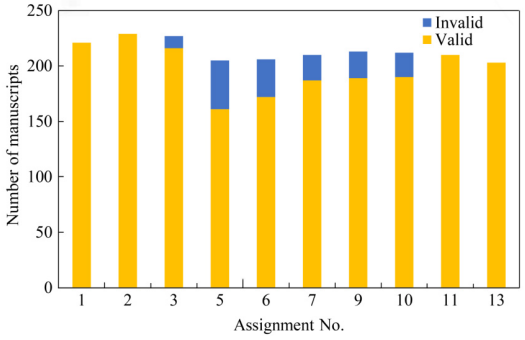


Fig. 4 General view of students' completion for each assignment.

m and n, as shown in Eq. (2). The total number of students is s, and the validity rate is shown in Eq. (3). In assignment No. 2, the manuscript code submission rate reached a summit of 94.2%, and all of the manuscript codes submitted proved to be valid. Assignment No. 5 shows the lowest submission rate and 66.2% validity.

$$p = n - m \tag{2}$$

$$\eta_p = p/s \tag{3}$$

Additionally, we collect and structure some other indicators: the pass rate of the public test suite in each training cycle, the general scores distribution in the peer code review process, and the OOCourse site visits for each training assignment. For each student's manuscript code, the lines of code and reported bugs are collected for statistical purposes. To better describe the students' learning outcome, we further collect the score data for all students in the preceding course, *Data Structures*, and make a detailed comparison of the two courses.

In order to facilitate the analysis of the results, this study groups the students' performance data by assignments. Each assignment should be completed within one week. The students' final performances are mainly evaluated by the results of the common test on the public test suite and the competitive test on the personal test suite. In order to explicitly reveal the formative process of change in the students' programming abilities, we separate our discussion of the statistical results of these two aspects.

4.2 Learning outcome as authors

Firstly, we discuss the results of the common testing in the peer code review process. The pass rate of public test cases in common testing is a direct indicator of the quality of a manuscript. We define η_i as the passing rate of public use cases for assignment i; η_i is calculated by Eq. (4):

$$\eta_i = \frac{c_i}{x_i \cdot z_i} \tag{4}$$

where c_i represents the general number of passed cases for assignment i, accordingly x_i represents the number of valid manuscripts and z_i is the total number of test cases in the public test suite. Figure 5 shows the overall performance of students in the public testing stage. Considering that programming assignments No. 14 and No. 15 focus on writing program specifications and are not included in the peer code review process, we collect the data on the first ten assignments. As can be seen from the figure, from assignment No. 6 onwards, the overall pass rate is increased with training tasks formatively implemented.

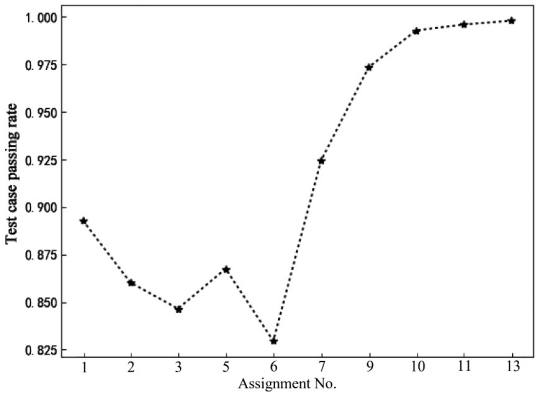


Fig. 5 Public use case passing rate for each assignment.

Besides the overall analysis of the students' public test cases pass rates, we analyze the public test cases pass rate from the individual perspective. Figure 6 shows the individual use case pass rate of students in the public test phase. As can be seen in the diagram, those students whose pass rates are greater than 95% make up 49.8% of all valid submissions, with a peak of 64.7% and a nadir of 28.7%. The result shows that a large quantity of student's manuscript code is able to achieve a reasonable pass rate on the public test suite. At the beginning of the course, with the difficulty of homework increasing, the pass rate of the students decreased slightly. Through a half-semester of programming training, the students' programming abilities are gradually improved, reflected by a continuous rise in the public use cases pass rate.

This study also analyzed and discussed the results of competitive testing for assignment Nos. 1–3, 5–7, and 9–11, in which testing by personal test suite is required. According to the result of each assignment,

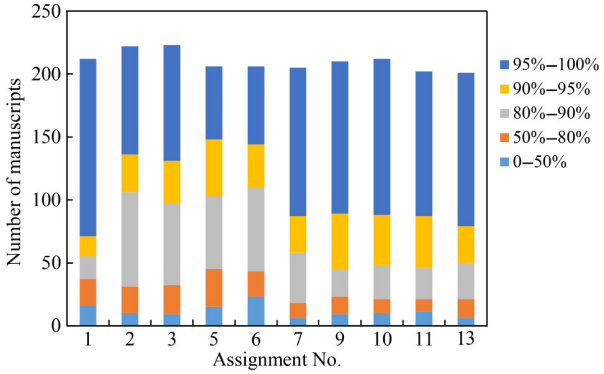


Fig. 6 Test cases passing rate for individual.

we classify students into different levels according to the grades they are achieving in competitive testing. The rules for the classification are shown in Table 4. For the classification rules, g represents the performance of the sample of students to be assigned: students in Level 1 have the strongest programming ability, while students in Level 4 have the weakest. Figure 7 shows the distribution of manuscripts at different levels for each assignment in competitive testing. As the assignments continued, the number of the students who submit valid manuscripts at a low level declines, while the number of students who submit valid manuscripts at a high level gradually rises after the fourth assignment. We can conclude from this that the students' programming ability levels continue to improve after three training assignments.

4.3 Formative assessment of project performance for students at different learning levels

In order to better analyze the changes in levels of programming ability by students throughout the semester, this study grouped all assignments into three learning cycles. In each learning cycle, there were three application development and peer code review tasks, plus a technical blog post. We analyzed the students' changes in level of programming ability per learning cycle.

This study defined the change in level of programming ability as the proportion of students

Table 4 Grading rules for ability level of students.

Grade	Rule
LEVEL1	$g \geqslant (\text{MaxScore} + \text{AvgScore})/2$
LEVEL2	$(\text{MaxScore} + \text{AvgScore})/2 > g \geqslant \text{AvgScore}$
LEVEL3	$\text{AvgScore} > g \geqslant (\text{MaxScore} + \text{AvgScore})/2$
LEVEL4	$g < (\text{MaxScore} + \text{AvgScore})/2$

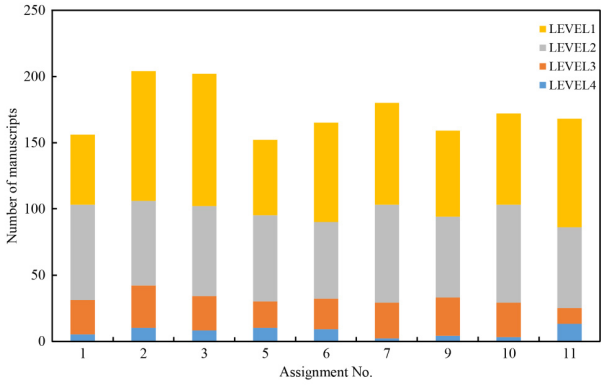


Fig. 7 Distribution of manuscript in competitive testing.

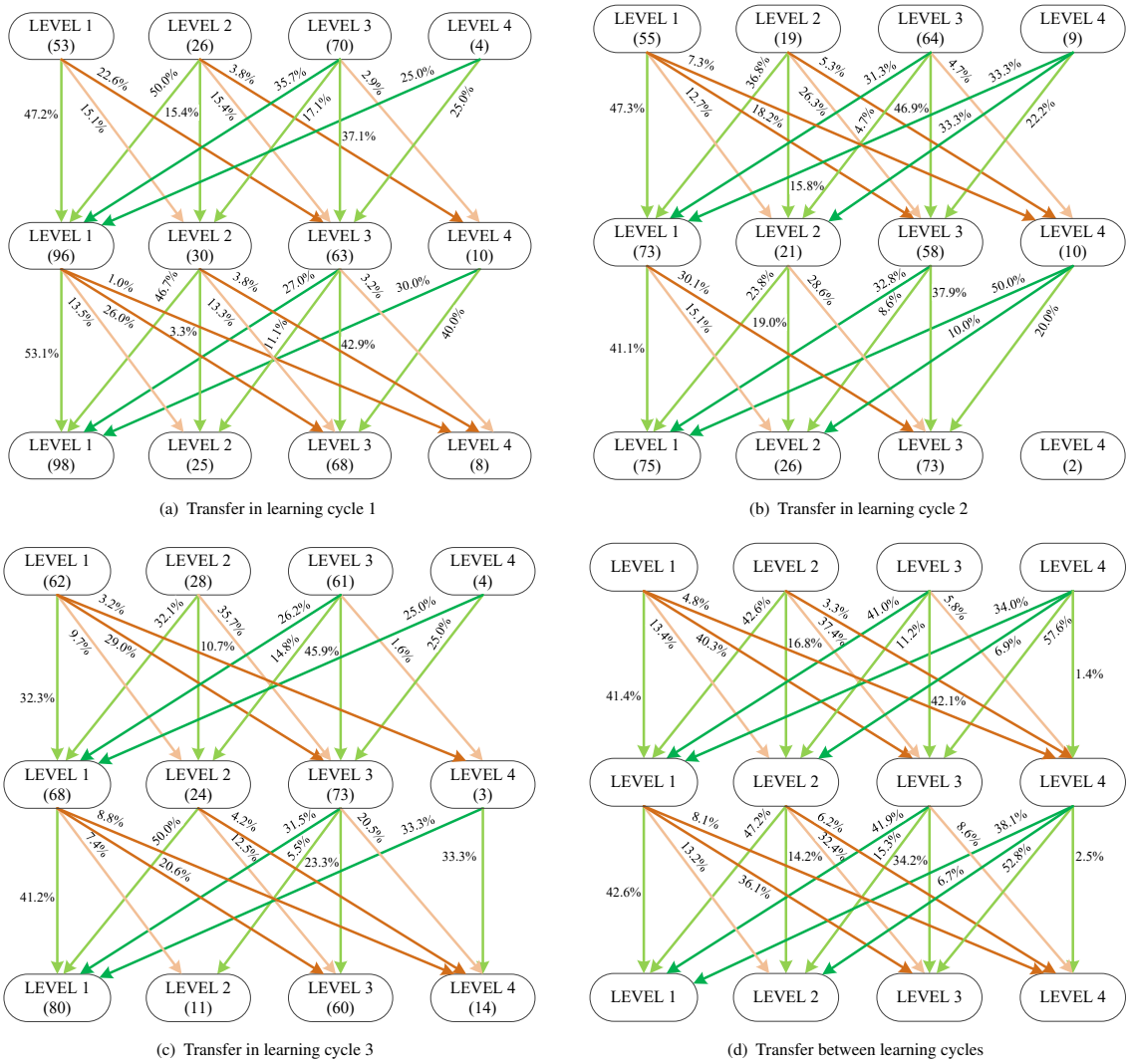
at a certain level transferring to another level in the following assignment. Therefore, for an assignment A_k, if m students are divided into level i ($i = 1, 2, 3, 4$), n of them are divided into level j in assignment A_{k+1}, we calculate the programming ability rate from level i to level j across assignments A_k and A_{k+1} as n/m.

The transfer of programming ability level by students in each learning cycle is described in Fig. 8. As shown in Fig. 8a, for the first learning cycle (assignment Nos. 1–3), in assignment No. 1, 53 manuscripts are regarded as at level 1, while the number of manuscripts in levels 2, 3, and 4 is 26, 70, and 4, respectively. In assignment No. 2, the number of manuscripts at level 1 notably rises to 96. Among the students who perform well (at level 1), 47.25% maintain their outstanding performance, while for the students whose manuscripts are at level 2, 50% of them transfer to level 1 in assignment No. 2.

The transfer of ability level in learning cycle 2 (assignment Nos. 5–7) is shown in Fig. 8b. Generally, it reflects a trend of transfer to a higher level as the assignments continued. The most attractive results occurred in the transfer from assignment No. 5 to assignment No. 6, in which the transfer rate to level 4 is zero. Excluding the 2 invalid manuscripts, none of the students who persisted in engagement with the course are graded into level 4. Learning cycle 2 takes place in the middle of the semester, and the results could be explained by the students' improved object-oriented programming abilities and a higher quality of manuscript code. With the deepening of the training, those students who start from a weak basis gradually transfer to a higher level of programming ability.

The same phenomenon can be observed in learning cycle 3 (as shown in Fig. 8c). The transfer regularity in learning cycle 3 is similar to the former two learning cycles. The majority of students transfer to the same or higher level as the training proceeds. We can see a formative improvement of programming ability throughout the training across the whole semester.

We also analyze the students' programming ability transfers between learning cycles. These results are shown in Fig. 8d. As for the transfer from learning cycle 1 to learning cycle 2, there are approximately 40% students who successfully transfer to level 1. The proportion keeps on rising to more than 40% when we focus on the transfer from learning cycle 2 to learning cycle 3. This indicates that the programming ability



(a) Transfer in learning cycle 1

(b) Transfer in learning cycle 2

(c) Transfer in learning cycle 3

(d) Transfer between learning cycles

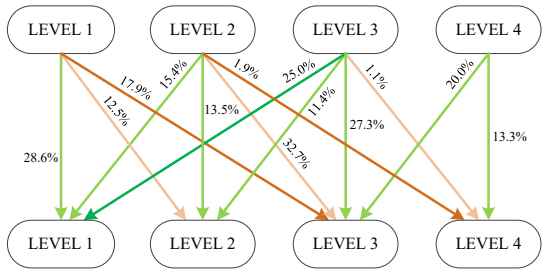
Fig. 8 Programming ability level transfer within learning cycle.

improved generally. A rather large number of students who start from a weak basis transfer to a higher ability level formatively.

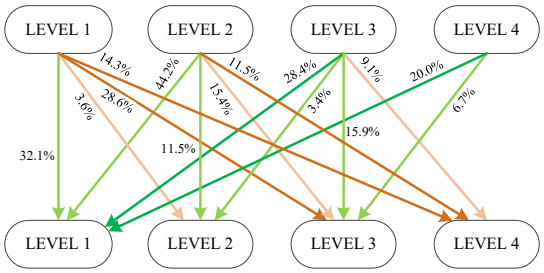
To make the conclusions more convincing, we collect the students' performance records from the compulsory preceding course, *Data Structures*, for comparisons. The same grading rules are applied in the *Data Structures* course for grouping student performance into 4 levels. Programming ability transfer across different levels was analyzed between the *Data Structures* course and our *Object-Oriented Design and Construction* course, which implements a formative peer assessment approach. Comparisons are made between the first assignment and the last assignment in the object-oriented course. The results are summarized in Fig. 9. By comparing the two figures, we arrive at the conclusion that students gradually improve their performance throughout the semester. At the beginning of the semester (assignment No. 1), among students who perform at level 4 in the *Data Structures* course, 20% maintain this level of performance or transfer to a higher level (Fig. 9a). However, with the completion of the peer review training process, 26.7% of them have transferred to higher level; especially, 20% of them reach level 1 (Fig. 9b).

5 Conclusion and Future Work

Peer assessment is useful since it simultaneously decreases teacher workload and improves the learning outcomes of students. To make the peer assessment results more reliable, we examined several critical issues, designed a formative peer assessment approach based on a peer code review model, and implemented it in a programming learning course. The model is



(a) The first assignment



(b) The last assignment

Fig. 9 Programming ability level transfer from data structure course to OOCourse.

well supported by an e-learning system OOCourse. Unlike former studies using surveys or questionnaires to measure a subjective evaluation of the effectiveness of the model, we collected objective data from the e-learning system based on peer code review outcomes. Statistical analyses were carried on the collected data set and interesting findings related to the transfer in ability level of students were uncovered. The results showed that the formative assessment approach was practical and successful.

The contribution of our work can be summarized into the following two aspects. Firstly, we proposed a formative assessment approach with a definition of essential roles and activities. Actual application data generated from the implemented course were collected to validate the effectiveness of the approach. The same methods could also be adapted for application in other programming language courses. Secondly, we provided useful and effective methods to reduce the risk of unreliable factors. Our work can provide inspiration for further rigorous models and solutions.

However, there are several aspects of our study that could be improved or expanded upon. Firstly, some prior work has suggested a multi-peer review as an extended model for peer assessment. However, the effectiveness of such model has only been validated by surveys and questionnaires, where the learning outcome is evaluated subjectively. We plan to extend our model by adapting it to multi-peer review and test this adaptation with real data. Additionally, student behaviors in the peer code review process could be regarded as gaming behavior, and their engagement could be explained in terms of competition and cooperation. Therefore, the formative assessment process might be improved by using a dynamic game theory model that includes awarding and punishing strategies. If the parameters of award and punishment in the review process are properly adjusted, we speculate that students will have more incentive to write manuscript code carefully and review code responsibly. This speculation will be examined in future research and practice.

References

[1] M. Zundert, D. Sluijsmans, and J. Merrinboer, Effective peer assessment processes: Research findings and future directions, *Learning and Instruction,* vol. 20, no. 4, pp. 270–279, 2010.

[2] S. D. Thanga, Impact of formative assessment: Comparative analysis on the performance of bachelor of science in education students, presented at the National Meeting of JWS, Tokyo, Japan, 2015.

[3] Z. Wang, B. Fan, X. J. Zhao, H. W. Yue, and A. M. Klushin, Integrating formative and summative assessment, *Oecd Education Working Papers,* vol. 28, no. 58, pp. 10–11, 2011.

[4] Y. Q. Wang, H. Li, and Y. Liu, Assessment of programming language learning based on peer code review model: Implementation and experience report, *Computers and Education,* vol. 59, no. 2, pp. 412–422, 2012.

[5] I. Cabrera, J. Villalon, and J. Chavez, Blending communities and team-based learning in a programming course, *IEEE Transactions on Education,* vol. 99, pp. 1–8, 2017.

[6] M. Bernhart and T. Grechenig, On the understanding of programs with continuous code reviews, presented at the IEEE International Conference on Program Comprehension, San Francisco, CA, USA, 2013.

[7] M. E. Fagan, Advances in software inspections to reduce errors in program development, *IBM Systems Journal,* vol. 15, no. 3, pp. 182–211, 1976.

[8] P. C. Rigby and C. Bird, Convergent contemporary software peer review practices, presented at the Joint Meeting on Foundations of Software Engi neering, Saint Petersburg, Russia, 2013.

[9] C. Hundhausen, A. Agrawal, and M. Trevisan, Integrating pedagogical code reviews into a cs 1 course: An empirical study, *ACM SIGCSE Bulletin,* vol. 41, no. 1, pp. 291–295, 2009.

[10] X. S. Li, Using peer review to assess coding standards: A case study, presented at the Frontiers in Education Conference, Las Vegas, NV, USA, 2007.

[11] C. D. Hundhausen, A. Agrawal, and P. Agarwal, Talking about code: Integrating pedagogical code reviews into early computing courses, *ACM Transactions on Computing Education,* vol. 13, no. 3, pp. 1–28, 2013.

[12] S. Sripada, Y. R. Reddy, and A. Sureka, In support of peer code review and inspection in an undergraduate software engineering course, presented at the IEEE Conference on Software Engineering Education and Training, Florence, Italy, 2015.

[13] Y. Q. Wang, Y. W. Liang, and Y. Liu, A multi-peer assessment platform for programming language learning: Considering group non-consensus and personal radicalness, *Interactive Learning Environments,* vol. 23, no. 6, pp. 2011–2031, 2015.

[14] T. Wang, X. Su, and K. Wang, Ability-training-oriented automated assessment in introductory programming course, *Computers and Education,* vol. 56, no. 1, pp. 220–226, 2011.

[15] Y. Q. Wang, L. Hang, Y. Sun, and Y. Jie, Learning outcomes of programming language courses based on peer code review model, presented at the International Conference on Computer Science and Education, Singapore, 2011.

[16] M. Matthias, Two controlled experiments concerning the comparison of pair programming to peer review, *Journal of Systems and Software,* vol. 78, no. 2, pp. 166–179, 2005.

[17] H. Ponbarry, W. L. Packard, and S. John, Expanding capacity and promoting inclusion in introductory computer science: A focus on near-peer mentor preparation and code review, *Computer Science Education,* vol. 27, pp. 1–24, 2017.

[18] B. M. Corsatea and S. Walker, Opportunities for moodle data and learning intelligence in virtual environments, presented at the IEEE International Conference on Evolving and Adaptive Intelligent Systems, Douai, France, 2015.

Qing Sun is a lecturer in the School of Computer Science and Engineering, Beihang University, China. She received the PhD degree from Beihang University in 2017, MSc degree from Beijing Normal University in 2008, and BSc degree from Beijing Normal University in 2005. Her area of research covers education data mining, information management, machine learning, and information systems.

Wenge Rong is an associate professor at Beihang University, China. He received the PhD degree from University of Reading, UK, in 2010, MSc degree with distinction from Queen Mary College, University of London, UK, in 2003, and BSc degree from Nanjing University of Science and Technology, China, in 1996. He has many years of working experience as a senior software engineer in numerous research projects and commercial software products. His research area covers machine learning, natural language processing, data mining, service computing, enterprise modelling, and information management.

Ji Wu received the PhD degree in 2003 from Beihang University, and is currently a faculty member in the School of Computer Science and Engineering, Beihang University. He has won the first-class Teaching Achivement Award by Beijing Municipal Education Commission in 2017. His research interests lie on modeling and verification of safety criticial system and software.

Wenbo Liu is a master student in Department of Computer Science and Technology, Beihang University, Beijing, China. He received the BSc degree from Tianjin Polytechnic University in 2015. His area of research covers machine learning and information systems.