Choosing a Better Moment to Assign Reviewers in Peer Assessment: The Earlier the Better, or the Later the Better?

Yanqing Wang
School of Management
Harbin Institute of Technology
Harbin, Heilongjiang 150001, China
yanqing@hit.edu.cn

Haoran Wang
Northeast Yucai Foreign Language School
Shenyang, Liaoning 110179, China
728999801@qq.com

Christian Schunn, Emily Baehr
Learning Research Development Center
University of Pittsburgh
Pittsburgh, PA 15260, USA
{schunn, ecb42}@pitt.edu

ABSTRACT

Peer assessment continues to be a topic of interest in the educational arena for decades, especially with the popularity of MOOC in recent years. However, the reviewer assignment moment, which may affect the process efficiency and learning proactiveness reward, lacks appropriate consideration. In this study, we propose three reviewer assignment algorithms, *post-assignment*, *pre-assignment*, and *submission-queue*, integrated with the educational peer code review system developed for the purpose of programming language learning. We compare and analyze the algorithms performance qualitatively according to *processing efficiency* and *proactiveness reward to learners*. In order to measure the three algorithms quantitatively, we carry out one preliminary investigation involving students in a programming course at a Chinese university. The results indicate that *submission-queue* has predominant advantages in both proactiveness reward and process efficiency relative to the other two. Moreover, data reveals that active students are more sensitive to reviewer assignment moment algorithms than the inactive ones.

Keywords

peer assessment; peer code review (PCR); reviewer assignment moment (RAM); submission queue; proactiveness reward; processing efficiency

1. INTRODUCTION

Peer assessment has become more relevant and has gathered increasingly more attention in education, as it fits the self-directed and collaborative learning processes [1]. Newman and Taylor defined the *reviewer assignment time* as the time taken to assign a reviewer to a paper and to record the assignment by conference organizers [2]. However, the *reviewer assignment moment* (RAM), which may affect the process efficiency and proactiveness reward to learners, has not been concerned yet. In previous researches on academic peer review, only a few studies have gone beyond performing an optimized assignment of manuscripts (or proposals) to reviewers in consideration of the quality assurance. In the studies of peer code review (PCR), scholars are mainly focusing on its quality assurance, learning outcomes, or sharing of successful instruction experiences. However, in our experience, we find that choosing an appropriate RAM does not only improve the efficiency of peer review process, but also stimulates the students' learning. Thus, in this study, setting a PCR software as the context, we concentrate our attention in answering two important questions: (1) what is the most appropriate moment to assign reviewer? and (2) what benefits can we obtain from applying the most appropriate reviewer assignment algorithm?

Since 2004, our team has been researching on PCR and we have been applying techniques for this end in two formal university courses: *C Programming* and *Object Oriented Programming in Java*. An e-learning information system dedicated to PCR in programming language learning, EduPCR, was developed and implemented by our team from 2007 for programming language learning and for e-learning research.

During the last few years, we have analyzed the output data from EduPCR to study the students' learning behavior. The system has inspired us to explore the students' e-learning data in different areas: their competence in coding, learning attitude, compliance with coding standards and the capability of following schedules [3]. Along with the previous studies, the RAM addresses the main concern of this study. In order to improve both students' activeness and efficiency in learning a programming language, we have tried a series of reviewer assignment algorithms relevant to RAM. This study seeks to find scientific evidence in different reviewer assignment algorithms, from perspectives such as *process efficiency* and *proactiveness reward to learners*.

The paper outlines as follows. Three RAM algorithms are proposed in section 2. Section 3 analyses their process efficiency and proactiveness reward qualitatively. These algorithms are studied comparatively in section 4. Discussions are made at last.

2. THREE RAM ALGORITHMS

So far, we have applied three practical reviewer assignment algorithms in the EduPCR system. Since every student plays the roles of reviewer and author, the final reviewer assignment result will build up one or several circles, namely, *review rings*.

2.1 Post-assignment

Post-assignment is a matured algorithm defined in peer review literature many years ago [4], for assigning resources or tasks after a specific phase. A networked peer assessment system for secondary science education utilized it [5]. In this study, we define *post-assignment* as "assigning reviewers after all students have submitted their source code".

We develop our own *post-assignment* strategy in EduPCR. With it, the assignment action is completed when all participants have submitted their source code or the stage deadline is reached. *Post-assignment* is the easiest approach of reviewer assignment for both, teachers' implementing and students' understanding. However, the disadvantage of *post-assignment* is that all the students who have submitted the source code prior to the deadline have to wait for the ones behind; if a single student fails to finish his/her source code, the rest of the students cannot start their review. Undoubtedly, *post-assignment* has a low efficiency rate. It is important to point out that there is no need to consider exceptional situations since all the students who are assigned as reviewers have finished and submitted their source code.

2.2 Pre-assignment

When the low efficiency of *post-assignment* was realized, *pre-assignment* became an alternative assignment method. It is a common algorithm widely applied to solve schedule problems; for instance, Tànfani & Testi proposed a *pre-assignment* algorithm to solve the Master Surgical Schedule Problem [6]. *Pre-assignment* is also used in educational peer review to achieve a certain goal in specific research [7].

Following these studies, we created our own *pre-assignment* algorithm in EduPCR. We define this algorithm as "reviewer assignment is finished the moment a programming task is released by teacher". Since every student can begin the review work as soon as the source code is submitted, this algorithm seems to be more efficient than *post-assignment*.

However, the *pre-assignment* algorithm is not without flaws. Due to the random review assignment in advance, eventually a reviewer has to experiment long waiting times for the manuscript of the assigned author. Additionally, one situation that often occurs is when one or more students miss the deadline of submitting their source code. For an instance in 1-to-1 assignment, see (a) of Figure 1, the student #2 who fails to submit source code on time will affect his/her two peers: the assigned reviewer #1 has no code to review and the author #3 has no reviewer.

In order to remedy this, the ideas of *compact algorithm* (see Figure 1) and *merge algorithm* (see Figure 2) are utilized so that the computer compacts the review ring by eliminating the students who do not submit their source code. When there is only one student left in a review ring, that student is labeled as a "lost" one, and will be merged to another review ring. Note that the merging position will be carefully chosen to make sure that the previous reviewer has not yet started his/her review work at the merging moment. In Figure 2, before merging, student #4 is the reviewer of student #5. If student #4 has not started the review work when the computer tries to merge, the merging can address this position, i.e. inserting student #3 between student #4 and student #5.

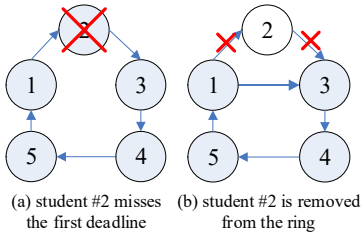

Figure 1. Illustration of the *compact algorithm*

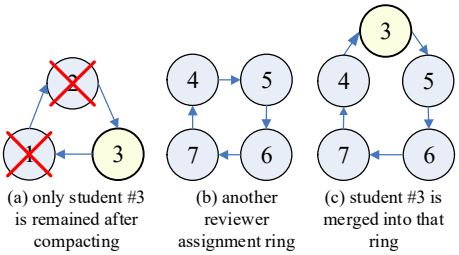

Figure 2. Illustration of the *merge algorithm*

When the system cannot find a position to merge a "lost" student into because all assigned reviewers have started (or finished) their review work, the system will collect all the "lost" students in all of the review rings and build up a new one. If there is only one "lost" student incapable of being merged, the system will notify with the message "You can revise your program upon your own idea since system fails to assign a reviewer for your work".

2.3 Submission Queue

Making use of the well-known scheduling strategy "first come, first served" in service policy or queuing law [8, 9], an "first submit, first assigned" algorithm (namely, submission queue), was implemented to overcome the problems brought by the *post-assignment* and *pre-assignment* algorithms. With the timer function, the computer checks who has submitted the source code at a predefined time interval (i.e. every 30 minutes). When the time is reached, the computer scans the number of students who have submitted their source code in the previous interval (let this variable be X), and the number of students who have not submitted their source code (let this variable be Y). Then there are three possible cases, called *assignment conditions*, as follows:

Case 1: $X<3$. If the source code deadline has not been reached, no action is needed. Otherwise, if $X=2$, the two students build up the smallest possible review ring (they are assigned as reciprocal reviewers); and if $X=1$, the *merge algorithm* is invoked (see Figure 2).

Case 2: $X\geqslant 3$ and $Y\geqslant 2$. The reviewer assignment will be performed among those X students.

Case 3: $X\geqslant 3$ and $Y=1$. In order to give a chance to the last student to submit the source code (we call him/her student L), we will let the penultimate student (we call him/her student W) who has submitted the source code, wait for L. In this case, our reviewer assignment will be performed among those X students, except for W. Even if $X=3$, the two students except for W can build up the smallest possible review ring. Finally, student W will not be assigned until student L submits his/her source code.

It is possible for only one student being unassigned by the first deadline. In *Case 2*, only one among all Y students submits his/her work before the deadline which will make the single student dangling. In *Case 3*, L has not submitted the source code when the 1st deadline is reached, student W will be dangling. To deal with dangling students, the *merge algorithm* (Figure 2), would be necessary. Theoretically, there is an additional scenario, in which all students in the last-built review ring have started their review work when the computer tries to merge. Despite having a very low probability, the system will notify the dangling student revise the source code according to his/her own understanding. Certainly, we can improve this algorithm by searching for an appropriate position in all review rings in the future.

2.4 Exceptional Cases and Solutions

Generally, most students can finish the 3 stages on time, but exceptional cases may occur occasionally, not only in source code stage, but in the reviewing and revising stage. The possible exceptional cases and their solutions are listed in Table 1.

3. EFFICIENCY AND PROACTIVENESS REWARD

From a managerial viewpoint, different algorithms may have different performances. The following section provides a capability analysis of the three RAM algorithms from two different perspectives: processing efficiency and proactiveness reward.

Table 1. The exceptional situations and corresponding solutions of three algorithms

Situations	Has influence on peers?		Solutions
fail to submit source code	*post-assignment*	No	The author who fails to submit source code is considered to quit the current program task. The assigned reviewer gets review marks automatically.
	pre-assignment	Yes	Compact a review ring by eliminating the students who fail to submit the source code. When the number of students in one review ring minimizes to one, the single student will be merged into another review ring.
	submission-queue	Yes	Only one student is still waiting to be assigned when the first deadline is reached. The student is merged into the last-built review ring.
fail to finish review	Yes		When the review deadline is reached, computer prompts the author to revise the program by himself/herself.
fail to revise	No		The corresponding marks of the author will be deduced.

3.1 Processing Efficiency

Whether a reviewer is active or passive, the possible starting point to review is critical because it bottlenecks the entire process. The assignment algorithm plays an important role in facilitating an active reviewer to start a review job as early as possible. The necessary conditions for starting a review work are:

(1) The author has submitted the source code;
(2) The reviewer has finished and submitted his/her own source code. To reduce plagiarism, the process constrains that a reviewer cannot review other's code before he/she submits his/her own source code;
(3) The reviewer assignment has been completed, that is to say that an author has an assigned reviewer who will review the source code. The assignment of a reviewer to an author depends on which of the three possible reviewer assignment algorithms is applied;
(4) EduPCR automatically informs the reviewer that the review process may start by short messages.

The starting time available for review is of great significance; among the four conditions, the 3rd might bottleneck the entire review process, hence it becomes fundamental to optimize the assignment moment.

It is easy to understand that the students can start the reviewing and revising process much earlier in *pre-assignment* and *submission-queue* than in *post-assignment*, which implies that the first two may be more efficient. Considering the individual activity, some students may submit their source code very quickly while others might submit their source code just before the first deadline. In *pre-assignment*, if the fastest and slowest students are assigned to a group to review each other, the fastest student will waste time waiting for the slowest student's source code. This issue is resolved in *submission-queue,* allotting for faster students to be assigned to fellow fast student.

Therefore, from the processing efficiency standpoint, we predict that *submission-queue* is the most effective and *post-assignment* is the least effective.

3.2 Incentive Effect to Learners

In *post-assignment*, all students have to wait until the last student submits his/her source code or the deadline, and then the assignment begins. Therefore, many students may choose to submit their source code until the deadline, since an early submission cannot trigger an early reviewer assignment.

In *pre-assignment*, the reviewer assignment is finished when the project (programming task) is released. A reviewer can start the review process only after his/her peer student (previously assigned) has submitted the source code. Students may submit their source code at their convenience, since an early or late submission will not accelerate or slow down the entire process much. If a student demands his/her source code to be reviewed, he/she must have submitted his/her own source code. On the other hand, if a student wants to review his/her peer's code, his/her peer must have submitted his/her own source code as well. The time interval between submission and review may be large but since there is no difference between submitting it early or late, the majority of students might submit their own source code as late as their peers do; which in turn becomes a submission close to the deadline.

However, in *submission-queue* we find two obvious advantages related to proactiveness rewards, relative to the other two assignment methods:

(1) Active students can maximize their learning pace as they wish, since whoever submits the source code first is assigned a reviewer therefore having their work reviewed first.

(2) *submission-queue* attracts students to join the active student circle. Normally, students who have good programming skills often submit their source code very early, and *submission-queue* can meet these students' time saving requirement. In addition, some students whose programming skills are limited may wish to be assigned in one review ring consisting of skilled students, in order to improve their abilities. Therefore, *submission-queue* can stimulate both skilled and limited students to submit their source code as soon as possible.

Hence, in terms of proactiveness reward, we predict that *submission-queue* may have predominant positive influence on students' leaning than the other two.

4. PRELIMINARY STUDY

The EduPCR system has been applied in the pedagogy of two courses: *C Programming* and *Object Oriented Programming in Java* for students majoring in *Information Systems* at a Chinese university. To understand the three assignment algorithms' effect, we conducted a comparative investigation on the course *Object Oriented Programming in Java* in one 23-student class. Because of lack of experience, we stored the review submission time in 12-hour (not 24-hour) format, losing some information and accuracy; the results might be affected to some extent.

4.1 Investigation Design

(1) There are ten tasks in this course. At an average interval of seven days, the teacher sets one task according to the lecture's content. Each task has three phases: submitting source code, reviewing peer's code, and revising source code. The average duration of a phase is about 2 days. For every phase, a deadline is set to control the process so that any activity that misses the deadline will be blocked.

(2) With each new task, all students are randomly divided into three groups to assure objectivity, thus, the sizes of the groups are about 8, 8, and 7 people. For every group, an assignment algorithm is randomly chosen among *post-assignment*, *pre-assignment* and *submission-queue*. We name each random group an *algorithm group* since each group is deployed with a different assignment algorithm. All students may start their review work only after they have received the review notifications sent by EduPCR.

(3) After all the ten tasks are completed, the time consumption is analyzed so that the efficiency of the three algorithms can be studied.

Since the time when students submit their source code is not greatly affected by the reviewer assignment algorithm and the revision submission time is indirectly affected by the assignment algorithm, we just focus on the time when the students submit their reviews. As previously mentioned, *post-assignment* algorithm starts the review process after the source code deadline, while *pre-assignment* and *submission-queue* algorithms can start the review process before the source code deadline. To compare the review efficiency of the three assignment algorithms, we take the difference between the review submission time of a reviewer and the source code deadline as the central statistic measure. This measure is named *wait time* (WT) of the reviewer; different reviewers have different WT values. Obviously, the smaller the WT value is, the more active the reviewer is. Then with *pre-assignment* and *submission-queue* algorithms, some reviewers' WT value is negative, which means that the reviewers finish their second stage task (i.e., review work) before the deadline of the first stage. During the implementation of EduPCR for years, we find that active students often finish their own source code and begin review work as soon as they get the "new review" notification, while some tardy students will not start each step until the deadline is approaching. Thus, the value of *average wait time* (AWT) can be used to measure the activeness of students in each assignment algorithm. AWT is the average value of several WTs, which has two cases: 1) *individual AWT* means the average value of one student's WTs within all tasks, which is used for clustering students into groups by their WTs; 2) *group AWT* is the average value of a group of students' WTs within all tasks. Each group has three group AWTs because three RAM algorithms are deployed.

4.2 Results from Data Clustered by Individual AWT

To eliminate the effect of different personalities, we cluster all students according to *individual AWT* within all tasks. Usually clusters generated from the clustering algorithm are not comparable. However, in this study, students are clustered by a single variable: *individual AWT*. Each cluster has a *group AWT* that is comparable with another *group AWT* of a different cluster so that we can order the generated clusters.

With K-Means in IBM SPSS Modeler, we cluster all students into four clusters according to *individual AWT* in ascending order, which means that the students in the first cluster spend the least amount of time finishing the review, while the ones in the fourth cluster spend the largest amount of time to do so. Moreover, we classify all the students as active and inactive; the students in the first two clusters of both investigations are considered active students, while those in the last two clusters are considered inactive. Finally, the clustering result is obtained, i.e. 5, 5, 7, 6 students in cluster 1 through 4 accordingly. Within each cluster, we compute the AWT values according to the *algorithm group*. The AWT of the four clusters in the three groups is shown in Figure 3.

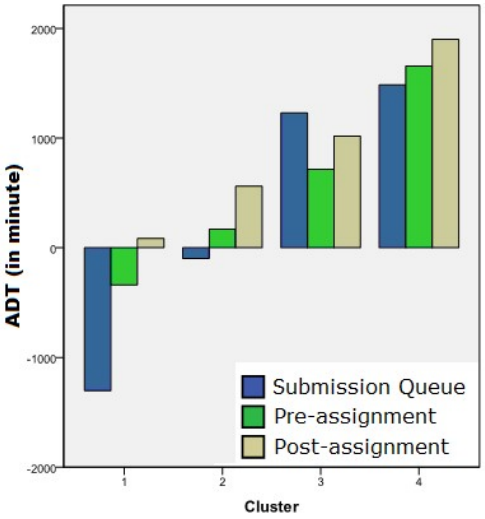

Figure 3. AWT distribution of the students

For each algorithm, the *group AWT* is increasing from cluster 1 to 4, meaning the activeness of students is decreasing correspondingly (see Figure 3).

Even though the database design defect might affect the accuracy of this study, the results are as follows:

(1) The processing efficiency prediction for the three algorithms is validated. Except for the *submission-queue*'s value of the third cluster in Figure 3 is a bit higher than their corresponding *pre-assignment* value, all the other values meet our prediction *(1=submission-queue 2=pre-assignment 3=post-assignment).*

(2) The proactiveness reward prediction for the three algorithms is validated. The time windows available to review with *pre-assignment* and *submission-queue* are the same. That is to say that, without the effect of behavioral factors, the *submission-queue* and *pre-assignment* AWT values of each cluster should not be greatly different. However, from the data shown in Figure 3, we can find that almost all the AWT values with *submission-queue* are much more efficient than those with *pre-assignment* because students do not need to wait the peer's source code submission for a long time after he/she submits his/her own source code. Moreover, the earlier a student submits his/her source code, the more chance he/she has of being assigned to a review circle with similarly active students. We do not mean that all active students are excellent, but the active students may like to study together with students of great learning enthusiasm. Thus, the result of the investigation demonstrates that the *submission-queue* does play the function of stimulating.

(3) The active students are more sensitive to the reviewer assignment algorithm. Within each cluster, the *submission-queue* and *pre-assignment* AWT values have difference, so do the AWT value of *pre-assignment* and that of *post-assignment*. We use the AWTs difference value to measure the students' sensitivity to the reviewer assignment algorithms. From Figure 3, we prove that the AWT differences in the first two clusters are much greater than those in the last two clusters, which indicates that the more active the students are, the more sensitive they are to the reviewer assignment algorithms.

5. CONCLUSION AND DISCUSSIONS

With the increasing requirement of peer assessment research, especially in the age of MOOC, the reviewer assignment becomes a pressing concern. The reviewer assignment research in the educational context, especially on the reviewer assignment moment, gets little focus.

From the perspective of RAM in the PCR context, we propose three algorithms comprising *post-assignment*, *pre-assignment*, and *submission-queue*. The preliminary investigation among one classes in an academic years reveal that *submission-queue* has a much higher processing efficiency and proactiveness reward than the other two algorithms. Moreover, we found that active students are much more sensitive to the application of reviewer assignment algorithms. Based on these results we discuss two main concerns:

(1) Do assignment pairs created with *submission-queue* algorithm promote fair feedback? This is an interesting issue addressed by some scholars with academic rigor. Actually we have to admit that proactiveness is the base of active learning, however, for the participants in peer assessment, being active is not the whole story. Fair feedback depends on the competence matching between peers, morality level of participants, adequate and appropriate training, quality assurance strategy by instructors, and so on. These are all challenging topics in the future research.

(2) Since *submission-queue* has predominant advantages relative to the other two, can it be taken as a universal algorithm in all scenarios? In our opinion, the choice of RAM algorithm depends on the context of each specific application. Even though *submission-queue* has more advantage over *post-assignment* and *pre-assignment*, these two algorithms also have practical values. For example, if the proactiveness reward and late submissions are not important, *pre-assignment* is easy to be applied and the entire process is simple to be managed. Similarly, if the proactiveness reward and completion time are not critical, *post-assignment* becomes a very suitable algorithm.

(3) Does *submission-queue* suit for MOOC or traditional classroom? We consider that the reviewer assignment moment in *submission-queue* always plays its role effectively in peer assessment no matter what the education context is. We find that *submission-queue* can stimulate students' learning enthusiasm and advance the task completion process especially when the students are inherently active. In traditional offline courses, the peer assessment process is the same as online process like in EduPCR so that *submission-queue* will suit for traditional classroom. More importantly, *submission-queue* tends to be more useful for assessing the performance of students in MOOCs. Although peer assessment greatly improves the grading efficiency in MOOCs, such assessment of open-ended assignments is still much complex and time-consuming with massive students in courses. If reviewers could start their review process as soon as possible, the peer assessment process will be largely advanced. Therefore, the processing efficiency and proactiveness reward of *submission-queue* will be more prominent in MOOCs.

ACKNOWLEDGEMENT

This work was partially funded by China Scholarship Council [201506125055], National Natural Science Foundation of China [71573065], and Online Education Research Foundation (QTone Education) of China's MOE [2016YB130].

Thanks to some students at School of Management, Harbin Institute of Technology, such as Miss Hang Li and Miss Xiaolei Wang, for their assistance of collecting data and analyzing the results.

REFERENCES

[1] Van Zundert, M., D. Sluijsmans and J. Van Merriënboer (2010). Effective peer assessment processes: Research findings and future directions. *Learning and Instruction, 20*(4), 270-279. doi:10.1016/j.learninstruc.2009.08.004

[2] Newman, W., & Taylor, A. (1999). Towards a methodology employing critical parameters to deliver performance improvements in interactive systems. In *Proceedings of INTERACT* (Vol. 99, pp. 605-612).

[3] Wang, Y., Li, H., Feng, Y., Jiang, Y., & Liu, Y. (2012). Assessment of programming language learning based on peer code review model: Implementation and experience report. *Computers & Education, 59*(2): 412-422. doi:10.1016/j.compedu.2012.01.007

[4] Mahoney, M. J. (1977). Publication prejudices: An experimental study of confirmatory bias in the peer review system. *Cognitive therapy and researc*h,1(2),161-175. doi:10.1007/BF01173636

[5] Tsai, C. C., Lin, S. S., & Yuan, S. M. (2002). Developing science activities through a networked peer assessment system. *Computers & Education*, 38(1), 241-252. doi: 10.1016/S0360-1315(01)00069-0

[6] Tànfani, E., & Testi, A. (2010). A pre-assignment heuristic algorithm for the Master Surgical Schedule Problem (MSSP). *Annals of Operations Research*, *178*(1), 105-119. doi:10.1007/s10479-009-0568-6

[7] Cho, K., & MacArthur, C. (2010). Student revision with peer and expert reviewing. *Learning and Instruction, 20*(4), 328-338. doi:10.1016/j.learninstruc.2009.08.006

[8] Chen, Y. (1999). Banking panics: The role of the first-come, first-served rule and information externalities. *Journal of Political Economy, 107*(5), 946-968.

[9] Wang, J., Cao, J., & Li, Q. (2001). Reliability analysis of the retrial queue with server breakdowns and repairs. *Queueing Systems*, *38*(4), 363-380. doi:10.1023/A:1010918926884