# Two Peers are Better Than One: Aggregating Peer Reviews for Computing Assignments is Surprisingly Accurate

Ken Reily, Pam Ludford Finnerty, Loren Terveen
University of Minnesota, Department of Computer Science
4-192 EE/CS Building, 200 Union St SE, Minneapolis, MN, USA 55455
{kreily, ludford, terveen}@cs.umn.edu

## ABSTRACT

Scientific peer review, open source software development, wikis, and other domains use distributed review to improve quality of created content by providing feedback to the work's creator. Distributed review is used to assess or improve the quality of a work (e.g., an article). However, it can also provide learning benefits to the participants in the review process. We developed an online review system for beginning computer programming students; it gathers multiple anonymous peer reviews to give students feedback on their programming work. We deployed the system in an introductory programming class and evaluated it in a controlled study. We find that: peer reviews are *accurate* compared to an accepted evaluation standard, that students prefer reviews from other students with *less experience* than themselves, and that participating in a peer review process results in *better learning outcomes*.

## Categories and Subject Descriptors

K.3.1 [**Computers and Education**]: Computer Uses in Education—*Collaborative learning*

## General Terms

Design, Experimentation

## Keywords

collaboration, peer review, education

## 1. INTRODUCTION

When one of Wikipedia's thousands of volunteer editors examines a recently changed article, that volunteer improves the overall quality of the work by participating in a distributed review process. Domains such as scientific peer review, industrial code review, open source software development, and information filtering on sites like Slashdot, Digg, or reddit also employ distributed review. We posit that the concept may improve education, too: student work (assignments) may be distributed to a set of reviewers who assess the work and provide formative feedback to the student. In an educational setting, another – and perhaps, central benefit – of distributed review is enhancing student learning. This paper systematically evaluates the use of distributed peer review in an in an introductory computer programming class.

Prior work has explored this subject. However, previous study of distributed review has left critical questions unanswered. First, there have been no controlled studies demonstrating the benefits of distributed peer review in introductory programming classes. Second, using distributed peer review in an educational setting raises a number of crucial challenges, such as preserving student privacy and avoiding "rogue" (e.g., retaliatory or collusive) reviews. This paper examines these important issues. We developed a system to facilitate the distributed review process for an introductory programming class. We then conducted a controlled, quantitative study to determine if distributed peer reviews are accurate and effective, to determine the impact the various activities (reviewing, being reviewed) had on learning outcomes, to assess the acceptability of peer review to students, and to identify the extent to which problems like rogue reviews occurred.

Today's educators face the challenge of delivering high-quality, individualized feedback to increasingly large classes. The traditional approach to this problem – using the same teaching and assessment techniques and requiring staff to cover the extra work – does not scale. This challenge forces us to look for new instructional and assessment techniques: hence, our interest in peer review.

Distributed peer review differs from traditional group programming projects because it uses a different time, different place learning environment. This paradigm may prove beneficial because computer programming assignments and long papers lack a structure conducive to same place, same time group learning; in fact, they have been identified as among the worst possible group assignments [28]. We view distributed review as an alternate form of collaborative classroom activity. Since it is used successfully in a variety of real world contexts, we posit that it may work well in a classroom setting.

We organize our work around the following research questions:

1. **Accuracy of Peer Reviews:** How accurate (relative to an accepted standard) are peer reviews in a computer programming class? This question is particularly


*GROUP'09,* May 10–13, 2009, Sanibel Island, Florida, USA.

important since the "peers" doing the reviews will not have advanced domain knowledge. *We found that peer reviews are accurate, that more reviews increases accuracy, and that more sophisticated aggregation algorithms help even more.*

2. **Review Effectiveness:** Do students consider peer reviews to be effective (helpful)? *We found that students generally found peer reviews effective, and that they preferred peer reviews from students who had less programming experience than they.*

3. **Impact on Learning Outcomes:** How does participating in peer review – whether writing or receiving reviews, or doing both – affect learning outcomes? *We found that reviewing other students' assignments resulted in significantly higher performance in the course.*

In the remainder of this paper, we discuss related work, describe our peer review system, discuss our experimental design, present and discuss our results in detail, and close with a brief summary.

# 2. RELATED WORK

This section presents related work in 2 parts: related research and existing system implementations.

## 2.1 Related Research

Previous research examines distributed review in a number of domains, including online communities. Lampe et al [21] demonstrate the basic utility of distributed review on Slashdot: they explore distributed comment moderation and find that distributed reviewers promote high quality comments and filter low quality ones. Cosley et al [6] study a more specific facet of online community distributed review: they examine the timing of the review process and conclude that the end result of post-review is just as good in terms of quality as pre-review. Like Lampe et al [21], we study the general utility of distributed review, but we do so in a classroom setting rather than in an online community. We take a broader view than Cosley et al [6] as we look at the end-to-end review process, rather than focusing the effects of review timing.

A number of researchers study the benefits of peer review in the classroom. Topping presents a comprehensive survey of peer review techniques in higher education [26], while Bloom et al. [2] and Krathwohl et al. [20] posit that because evaluation and criticism are high level skills in Bloom's Taxonomy, students who participate in a peer review process achieve a high order educational goal. Hamer [17] summarizes benefits to using peer review in the classroom, including increasing the quantity and timeliness of feedback, exposing students to different techniques and styles, and encouraging reflection on the course objectives through the review task.

Sullivan and Collofello [25], [5] and Gehringer [9], [10], [11], [12] also examine peer review in the computer programming classroom. While Gehringer cites a number of benefits to using peer review as a classroom exercise, neither he nor other researchers perform a systematic quantitative evaluation of the accuracy (with respect to a standard) or of learning benefits of peer review as we do. Trytten proposes a design for team peer code review [29] and proposes that the structure of the review exercise is nearly as important as the exercise itself. We agree, and we incorporate this structured approach into our electronic system for facilitating the review process. Denning et al [7] examine *in-class* peer review for computer programming classes. They create a model for completing reviews quickly in a time-constrained class (a "same time, same place" model). We instead employ a "different time, different place" collaboration model that allows reviewers to complete their reviews at a time and place convenient for them.

One potential concern about using peer review in the classroom is shilling or rogue reviewers – those reviewers who, for a variety of reasons, give arbitrary scores regardless of submission quality. Hamer et al [17] propose an algorithm for automatically calibrating peer review scores, however they evaluate their algorithm on simulated data only. Here, we apply the algorithm to actual data.

In addition to the work specifically mentioned above, Anewalt [1], Hamer [16], Wolfe [30], Trivedi [27], Silva [24], Liu [22], and Gotel [14], [15] all present research that describes the design and implementation of systems and processes for peer review in computer programming courses. However, much of the previous work focuses on recounting lessons learned from the implementation of the systems and processes. Here, we design and conduct a controlled study designed to determine the accuracy of peer review, what makes a review effective, and the impact of the peer review process on learning.

Beyond distributed review, there are other methods for assessing the quality of work in a classroom setting. For example, Foltz [8] proposes a technique for automatically scoring essays using semantic analysis. However, it is not clear if this technique can be applied to computer programming or if it is widely used in the classroom.

## 2.2 Existing Systems

A number of web-based systems facilitate distributed review. MyReview (myreview.lri.fr), ConfTool (www.conftool.net), and Precision Conference (www.precisionconference.com) support peer review for academic conferences. Peer Grader (PG) was an early web-based system for submitting of computer programming assignments for peer review [10]. In a later iteration, PG added resources for teaching computer architecture [11]. Later, it evolved into Expertiza, which used student peer review to improve an unpublished textbook [13]. It is currently in classroom use at North Carolina State University.

We considered using one of the aforementioned review platforms for our research. However, they left key requirements unmet, for instance: (1) the existing classroom systems were domain independent and did not support programming specific concepts like test cases and code characteristics (see below for definitions). Since evaluating test cases was a requirement for our peer review system, we built our own. (2) We wanted assignment authors to provide structured feedback for a review; we did not find any existing tools that did this. (3) Many systems we evaluated are hosted at external web sites. Because we conducted our study in a classroom, we had to meet strict data privacy requirements, both legal (as specified in the United States Federal Educational Rights Privacy Act) and institutional (as specified by our University rules). Therefore, we decided to implement our own peer review application.

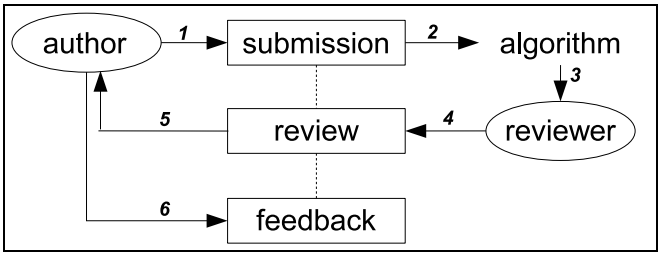


**Figure 1: Flowchart illustrating the submission and review process.**

# 3. PROCESS AND APPLICATION DESIGN

This section gives an overview of the application design and the process it supports.

Users (students) play two roles in the peer review process. *Submission authors* (hereafter called authors) create and submit solutions to programming assignments. *Reviewers* create structured reviews of assignments submitted by authors. In our study, students were assigned either, both, or neither of these roles.

The peer review process consists of the following steps (see Figure 1):

1. Students submit their assignments via our web-based submission application.
2. The application assigns each submission to be reviewed by the course Teaching Assistant (TA) and three other students using a randomization algorithm.
3. The application notifies all reviewers that the review assignments are available in the system.
4. The TA and student reviewers complete and submit their review by filling out a structured online form.
5. The application notifies the submission author after all reviews are complete and ready for viewing.
6. The submission author then rates each review and provides free-form text feedback to the reviewer(s).

The review assignment algorithm is a simple random algorithm with 2 obvious constraints: 1) students may not review their own submissions, and 2) no student may review the same submission more than once.

The reviewer provides feedback on the programming assignment through test cases and code characteristics.

**Test Cases.** Test cases measure the performance of the submission against the assignment requirements. For example, one requirement may be to sum two numbers. A test case for this requirement may invoke the sum function with the numbers 5 and 10 and verify that the result is 15. One requirement may give rise to many test cases. Our application requires reviewers to write a minimum number of test cases (initially 3; later increased to 5 as assignment complexity increased). However, reviewers were allowed to write more. For each test case, the reviewer indicated an outcome from the following choices: does not compile, fatal error/crash, incorrect result(s), or correct result(s). We used the rankings to measure performance on test cases.

**Code Characteristics.** Not all aspects of a computer program can be assessed via test cases. For example, determining whether an interface design is "good" or "bad" and whether documentation is sufficient require human judgment. Therefore, the application also lets reviewers rate the submission's code characteristics including formatting, effective use of conventions, documentation, interface design usability, and modularity.

For each code characteristic, the reviewer indicates his or her level of agreement with a statement about the characteristic using a 5-point Likert scale. For example, for rating documentation, the reviewer sees these statements: "The code is well documented (commented). The comments are written in the problem language and will be easy to maintain as the program changes over time. Comments are used judiciously."

The reviewer also can – and, in our study, was encouraged to – provide text comments for each test case and code characteristic. See Figure 2 for examples. We combined the outcomes of the test cases with the numerical rating of the code characteristics to produce a quantitative review score. The review in Figure 2 resulted in a score of 94/100. This score plays an important role in subsequent analysis; notably, it lets us compare student and TA reviews quantitatively.

**Author Feedback to Reviewer.** Upon receiving their review(s), the application required authors to give feedback on the review by rating and commenting on four aspects of the review: accuracy, helpfulness, reviewer knowledge, and fairness. These feedback ratings provide a mechanism for analyzing the effectiveness of different author/reviewer pairings. Figure 2 shows a typical completed review with feedback from the submission author.

# 4. EXPERIMENT DESIGN

We performed a controlled study to answer our research questions in the Fall of 2008. An introductory Information Systems class served as the backdrop for our study. The first author of this paper was the instructor for the course (which he had taught once previously.)

To review, we summarize our research topics: *Research Question 1*: we compare student reviews to those of the TA (the "gold standard"); *Research Question 2*: we measure review effectiveness (helpfulness); *Research Question 3*: we measure the impact of participating in peer review on learning outcomes.

We used a $2 \times 2$ experimental design. The two factors were *writing reviews* (yes or no) and *receiving reviews* (yes or no). Table 1 introduces names for each of the four resulting groups – G-CONTROL, G-REVIEW, G-RECEIVE, and G-BOTH – that we use throughout the paper. This experimental design let us test separately the effect of receiving and writing reviews as well as any added effect of doing both. We wanted the four groups of students each to be representative of the course as a whole in terms of incoming grade point average and prior programming experience. However, we did not want to prejudice the course staff and researchers about student performance. Therefore, we gave instructions for creating groups to our university's Office of Institutional Research, and they assigned students to groups.

51 students began the semester and 45 students completed it. This is a low drop-out rate for this course: for many students, this is the first course in their major (Information Systems), and some leave if they find it too challenging or not to their interest. Of the 51 initial students, 25 were in groups that received peer reviews (G-RECEIVE and G-BOTH) and 25 were in groups that wrote peer reviews (G-

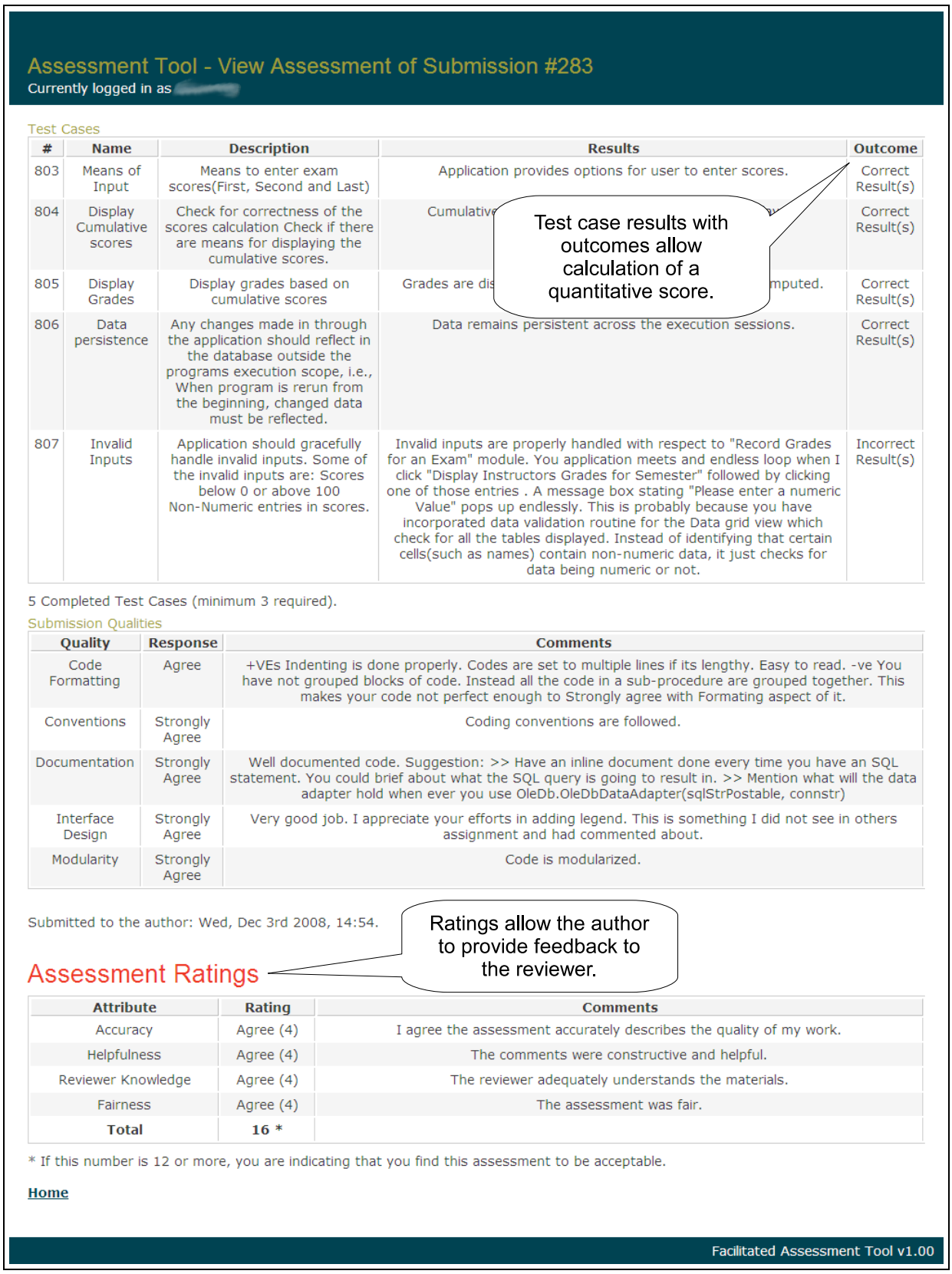


**Figure 2: A typical completed review as seen by the submission author. The author's ratings of the reviewer appear at the bottom. This review resulted in a quantitative score of 94/100.**

| Group | Receives? | Writes? | Initial # | Final # |
|---|---|---|---|---|
| G-Control | N | N | 13 | 13 |
| G-Review | N | Y | 13 | 10 |
| G-Receive | Y | N | 13 | 11 |
| G-Both | Y | Y | 12 | 11 |

**Table 1: Our 2×2 design for analyzing how the different learning activities (reviewing, being reviewed) impact learning outcomes along with the initial and final sizes for each group.**

Review and G-Both). Table 1 shows initial and final sizes for each group.

All students in the course used the review software to submit six programming assignments. Students in G-Review and G-Both each reviewed 1 to 3 submissions for each of the six assignments. Students in G-Receive and G-Both received up to 3 peer reviews for each assignment. Having up to 3 student reviewers for each submission gave authors multiple sources of feedback and let us test how using different number of peer reviews (one, two, or three) affected accuracy. Students in G-Control did not write reviews and did not receive peer reviews. However, like all students, they received reviews from the course Teaching Assistant (TA). *Authors did not know whether a review came from the TA or a fellow student.*

To measure the accuracy of student reviews, we used reviews from the TA as the "gold standard". The TA was a Computer Science Master's degree candidate with significant industry programming experience. Of course, TA grades are not a foolproof or an absolute measure of submission quality; nonetheless, they are the accepted standard for course work at our University and generally throughout higher education.

To measure student opinions about the process and to inform the quantitative results, students completed pre-study and post-study surveys. The pre-survey was the same for all participants. We prepared somewhat different versions of the post-survey for each of the four study groups.

To ensure fairness to the students, assignment scores were based solely on the TA's review. In addition, because our school has a median grading policy, final letter grades for the students were assigned with respect to their study group and not the class as a whole. This is because of the potential that review activities could lead to different learning outcomes for each group (which we report on below).

# 5. RESULTS

We organize the discussion of our results around our three research questions.

## 5.1 Accuracy of Peer Reviews

During the study semester, we collected 378 individual peer reviews for the six programming assignments. We analyzed the accuracy of the reviews under several aggregation schemes.

**Students rated submissions similar to the TA, but more harshly.** We calculated the quantitative score for each review and then compared the scores of the TA and student reviews. We calculated the score by giving the test cases a 75% weight and giving the code characteristics a 25% weight. If we correlate the TA score with each individual student score, we get a Pearson coefficient ($r$) of 0.679, which is marked correlation. 24.6% of student review scores (93 out of 378) were higher than the TA's score (for the corresponding author and assignment), 17.2% of student review scores (65 out of 378) were equal to the TA's score (for the corresponding author and assignment), and 58.2% of student review scores (220 out of 378) were lower than the TA's score. On average, student review scores were 2.6 points (out of 100) lower than the TA's score.

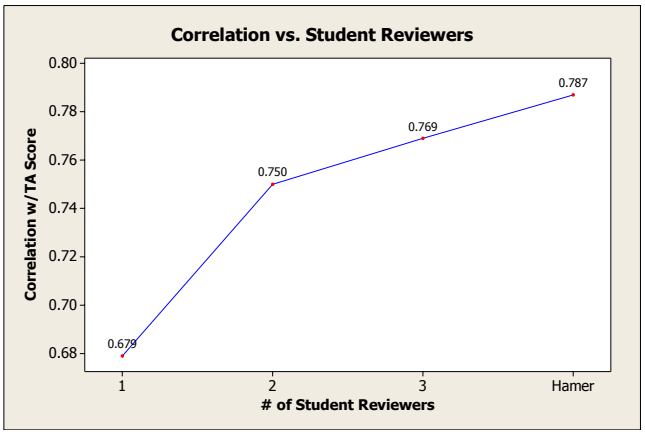


**Figure 3: The correlation of of student review scores to those of the TA increases as you increase the number of student reviewers. "Hamer" is the result of applying Hamer's calibration algorithm. Note the y axis does not start at 0.**

**Aggregating multiple student reviews improves accuracy.** On average, the correlation between the mean of any two student review scores and the TA's score (for the same assignment and author) was 0.750. Taking the mean of all three student reviews improved $r$ even further to 0.769.

**Smart aggregation improves accuracy even more.** A simple average of student review scores may not be the best technique. In addition, student reviewers could adopt a number of rogue strategies. For instance, they may give every submission a perfect score out of laziness, or they may be particularly aggressive in trying to "break" the author's program by giving it input they know will never be processed correctly. To avoid such problems, Hamer et al [17] designed an iterative algorithm for automatically calibrating peer review scores. Applying Hamer's algorithm to our data improves the correlation to 0.787. The improvement is minimal over the naive approach (3-student mean). However, Hamer's algorithm provides additional information about the quality of student reviews that can be used to provide an incentive for students to write good reviews. In particular, it computes weights that indicate each reviewer's overall agreement with the consensus of all reviewers. These weights may provide a mechanism for rewarding the most accurate reviewers. (Note that reviewing systems for academic conferences sometimes present such data to reviewers as a self-assessment aid.) The algorithm also provides a mechanism for detecting potential plagiarism.

Figure 3 summarizes the correlation improvement as we aggregate more student scores and then apply Hamer's algorithm.

**Survey**. The survey provided additional insight into the accuracy of peer reviews. 15 of 21 student reviewers agreed that they had the competence necessary to review other students' work. In addition, 17 of 21 student reviewers indicated that they reviewed other students' work fairly. Students receiving the reviews seemed to agree: 15 of 22 students who received peer reviews indicated the other students' reviewed their assignments fairly, and 13 of 22 students who received peer reviews indicated that the other students had the competence necessary to do the reviews. Only 4 of the 22 students who received peer reviews indicated that, based on their experience in this course, they would not want to take another course that used peer review.

### 5.1.1 Review Accuracy Discussion

Our results showed that multiple peer reviews were accurate proxies for an individual expert review, even when the peers were novices. Previous research found that multiple peer reviews can even outperform individual expert review in certain situations [3]. Using peer reviews as a means to improve work quality and enhance learning seems relatively unproblematic. And the thought that peer reviews could be used to reduce the amount of work required from a teaching assistant is appealing. However, the last step – using peer reviews in computing course grades – raises some significant challenges, notably: rogue (dishonest or poor quality) reviews and comprehending or questioning grades. We now discuss this issue.

**Taming the rogue review: incentives for honest reviews**. In any system where participants review each other, there are a number of factors that may lead to untrustworthy or "rogue" reviews. Using peer reviews for grading purposes raises more factors. Here are the main reasons we consider:

1. *Retaliation* – participant X gives an undeserved negative review to person Y in response to a (perhaps deserved) negative review that person Y gave to person X (Resnick et al. studied this issue in EBay [23]).

2. *Collusion* – two participants agree to give each other high (and perhaps undeserved) reviews.

3. *Competition* – participants compete for some limited resource, e.g., grant money, acceptance at a highly selective academic conference, or (most relevant here) a top grade in a course that is graded "on the curve". The latter issue was at play in our work: our college limits the number of top grades (B+ and above) that may be assigned in a class. Thus, a student might reason that he or she could move "up the curve" by grading other students harshly. The same reasoning applies in academic peer review; conflict of interest policies and mechanisms like double-blind reviews are used to combat such temptations.

4. *Laziness* – it takes effort to write a good review. A student might reason that the effort would be better spent on improving his or her programming assignments, and thus submit a shallow, ill-considered review.

We anticipated several of these situations and tried to create incentives to head them off. First, students who wrote reviews (i.e., groups G-Review and G-Both) were required to submit their reviews before receiving their assignment score. This was designed to ensure timely completion of reviews. Second, authors rated each review they received. While we did not use these ratings in computing reviewers' course grades, they could have been used for this purpose. This would be a strong incentive for writing honest and substantive reviews. Further, the quality of a review could be estimated objectively, e.g., based on the amount of text and the distance between its score and the average score of all reviews. Finally, to try to prevent "laziness" due to the effort of reviewing, we explicitly considered writing reviews as a course activity, adjusted the course workload accordingly, and communicated this to students.

In practice, we saw little or no evidence of rogue reviewing. A few students did express worries. Some reviewers said they were concerned that authors might retaliate if they (the reviewers) gave low scores. Four or five reviewers told the instructor that they suspected receiving retaliatory ratings in such cases. We did analysis to try to quantify whether retaliation was occurring; specifically, we correlated review scores and corresponding review ratings from the submission authors. The correlation was weak ($r = 0.216$). We also discovered several instances of collusion where pairs of students realized that they had received each other's assignments to review and then agreed to give each other positive reviews and ratings. Collusion could be prevented by a number of methods, such as an algorithm that provides for specifying a minimum transitive distance between author/reviewer pairs. This is a likely candidate for further research.

Despite little evidence of rogue review, this issue clearly requires more work if peer reviews are to be used in assigning grades. Algorithms (like Hamer's) can help identify rogue reviews. More importantly, inventive systems could prevent rogue reviews in the first place.

**Comprehending aggregated peer review grades.** Under a traditional grading system, when students don't understand or agree with their grades, they go complain to the TA or instructor. However, what should they do if their grade is based on some mechanical aggregation of scores from multiple anonymous peers?

Our post-survey illustrates that students are concerned about this issue. While only 3 of 22 students reported that they had problems raising grading issues with the TA, 10 of those same 22 students expressed concern that they would have no way to dispute or appeal peer review scores. In addition, students who received peer reviews disliked the idea of using them as a grade determinant: 15 of the 22 students disagreed with the statement, "Receiving reviews from 3 other students on my programming assignments would be an acceptable substitute to receiving an assessment from the TA."

Next, while TA assessments are not perfect, replacing or complementing them with a complex aggregation of peer reviews may not provide sufficient understanding of why they received a particular score. While students comprehend written feedback from the TA (e.g. "You lost 5 points because you did not do x"), receiving multiple peer reviews from peers along with an aggregate score may leave the students without a clear understanding of what caused them to lose points or what actions they can take on a future assignment to improve.

Explanation systems might improve the comprehensibility of grades based on peer reviews. Explaining decisions reached by computational systems in order to make them

more acceptable to people is a well-established research topic. Herlocker et al. [18] developed and evaluated a number of explanation interfaces for recommender systems. More relevant to our concerns is the work of Clancey [4]. He showed how it wasn't enough for a medical expert system to make accurate diagnoses; it also had to convince physicians that these diagnoses were right. It turned out that to compute convincing explanations, it wasn't enough to just describe the expert system's reasoning process; the reasoning process had to be re-designed so that it was comprehensible to people. Likewise, we will need explanations along with aggregation algorithms for peer review systems to be acceptable for classroom use.

## 5.2 Review Effectiveness

On the post-survey, 14 of 22 student authors (i.e., members of groups G-Both and G-Receive) reported receiving 3 or more effective reviews over the course of the study. To further understand review effectiveness, we carried out analysis (described below) based on the ratings authors gave to the reviews they received. Recall that section 3 detailed the rating criteria: accuracy, helpfulness, reviewer knowledge, and fairness.

Using peer review in a classroom setting raises the question of whether students – who are just learning the material – are qualified to review the work of other students. A reasonable intuition would be that the more students know, the better reviewers they would be. Students seemed to share this intuition: the survey results showed that students ranked "A reviewer with more experience than me" as the second most important factor in judging review quality. However, our results were not consistent with students' intuitions: in fact, students preferred reviews from other students with similar experience.

We computed these results using authors' ratings of reviews and the pre-survey. The pre-survey asked students to rate their own programming experience and to indicate the number of prior computer programming courses they had taken. We then divided all the review-ratings (i.e., authors' rating of reviews) into four sets, based on the relative self-reported experience of the author and reviewer: ratings of the reviews they received into four groups based on the relative experience level of the reviewer and author:

- **Author** Ratings where the author had more programming experience.
- **Equal** Ratings where the author and reviewer had equal experience.
- **Reviewer** Ratings where the reviewer had more more programming experience.
- **TA** Ratings of reviews by the TA.

An ANOVA shows a statistically significant difference in ratings between the four groups ($p < 0.01$). Follow-up pairwise Tukey tests show that all differences between groups were significant, with the exception of the Author and TA pair. Two major observations can be drawn from these results. First, students' self-reported preference for reviews from more experienced peers was not supported; in fact, students liked reviews from more experienced peers least of all. Second, students liked reviews from less experienced peers as much as reviews from the course TA. While the TA almost certainly had a significantly higher experience level than all students, note that the TA has received training on writing effective reviews and his or her continued employment at the school depends on doing the job effectively. Figure 4 shows the average ratings by group based on self-reported experience levels.

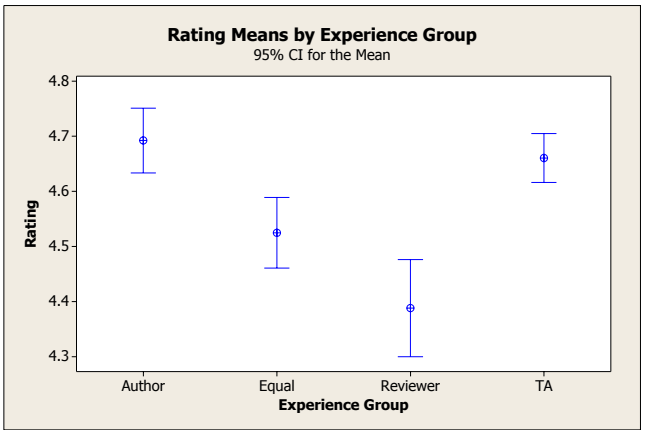


**Figure 4: Author ratings of reviewers were divided into groups based on self-reported experience levels. Reviewers with more experience than the author fared the worst. Note the y axis does not start at 0.**

However, we wondered whether students did not accurately assess their experience level on the pre-survey. Or, put another way, perhaps self-rated experience level doesn't correlate well with programming *knowledge*. Additional data analysis lets us investigate the effect of differential programming knowledge more objectively. The idea is to take final course performance (grade) as a measure of knowledge. We define $P_r$ to be the overall course performance of the reviewer and $P_a$ to be the overall course performance of the author. Then $\Delta P = P_r - P_a$ represents the difference in course performance between the reviewer and the author (a proxy for differential knowledge). A positive $\Delta P$ represents a case where the reviewer outperformed the author, and a negative $\Delta P$ represents a case where the author outperformed the reviewer. Figure 5 shows a regression of the review rating vs. this performance differential. While there are lots of data points ($n = 1412$) that are fairly scattered ($r^2$ is relatively low), the downward trend is obvious.

According to students, the first and third highest ranked factors contributing to an effective review were the quality and quantity of written feedback. Quality is subjective and thus hard to quantify. However, we did check whether the quantity of written feedback in a review correlated with the author's rating. It did not ($r = -0.073, p = 0.172$).

**Discussion.** Students' general intuition – that they want reviewers from more knowledgeable peers – did not correlate with their actual ratings of the reviews they received – they rated reviews from (apparently) less knowledgeable peers more highly. However, while we found this surprising, other research suggests that perhaps we should not have. Hinds identified a "cognitive handicap" that experts have when attempting to identify with the skill set of a novice [19]. Cho examined the impact of novice peer-based knowledge refinement in knowledge management systems [3]. In both cases,

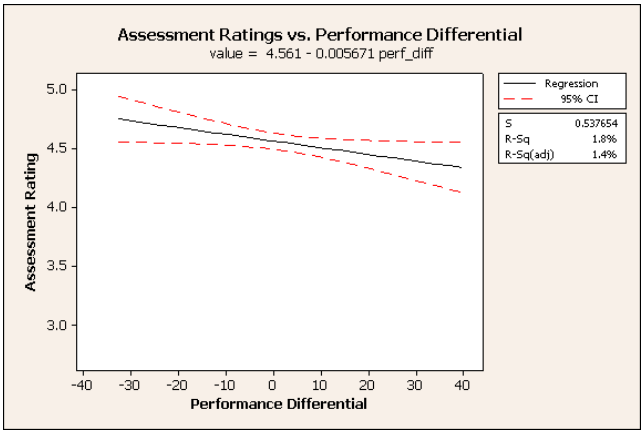


**Figure 5: A regression of review ratings vs. performance differential. A high performance differential indicates the reviewer significantly outperformed the author in the course. Note the y axis does not start at 0.**

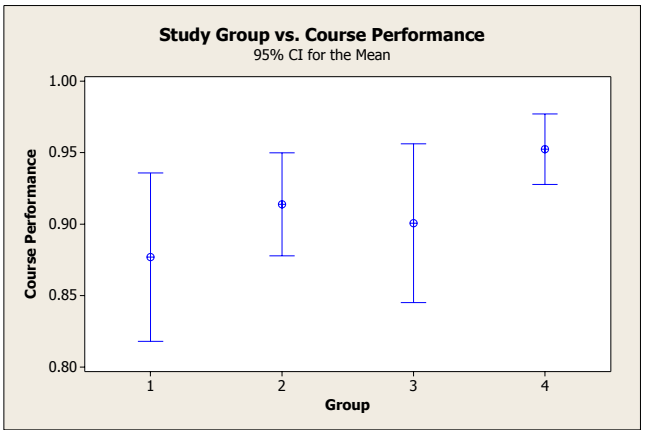


**Figure 6: An interval plot of course performance vs. study group. Group 1 is the control group (G-CONTROL) that did not participate in any peer review activities, while group 4 both wrote and received reviews (G-BOTH). Note the y axis does not start at 0.**

the researchers found fellow novices to be as good, if not better, than experts at performing certain tasks that involved assessing the quality or predicting the performance of other novices. Finally, Zhang et al [31] also speculate that expertise networks based on the users with highest absolute expertise may encounter some strain. They propose a "just better" model that balances the network using a hierarchical approach where users' questions are answered by other users with slightly more experience than themselves. This model might provide the basis for an improved classroom review system.

## 5.3 Impact on Learning Outcomes

To measure learning outcomes, we calculated an aggregate score (out of 100%) for each student across all assignments and exams. The average score for for the control group was the lowest of the four groups (87.7%); the average score for students who both wrote and received reviews (G-BOTH) was highest (95.2%). An ANOVA showed that the difference between the groups was not significant, although there was a strong trend ($p = .1$).

We also hypothesized that students who acted as peer reviewers would perform better in the course because (a) seeing other students' approaches to the same programming assignments would help reviewers learn other paths to a viable solution, and conjectured that this knowledge, which one cannot get from TA-grading only or being reviewed by peers only, would prove valuable. (b) Assessing other students' work requires significant cognitive effort: reviewers had to determine whether their peer's work was correct, and have some understanding of why it was right or wrong. We also thought this would boost learning. To test our hypothesis, we performed a t-test, comparing student performance in G-REVIEW and G-BOTH (reviewer conditions) vs. performance in G-CONTROL and G-RECEIVE (non-reviewer conditions). Reviewers (mean performance 93.406) performed better than non-reviewers (mean performance 88.78), and the differences were statistically significant (p = 0.04, t=2.1167, df 43, std err of difference 2.185).

We also wondered whether receiving feedback from peer reviewers would boost performance, so we performed a t-test comparing students who received peer reviews verses those who did not, but differences were not statistically significant.

Figure 6 shows an interval plot illustrating the learning outcomes for each group.

**Discussion.** The study survey offered further support for the idea that participating in peer review activities enhanced learning. Several students noted that the process of reviewing others' work was surprisingly informative, indicating that they would often look at the submissions they were assigned to review for ideas of how to improve their own programming. When asked what they learned by reviewing the work of other students, responses included:

- "Different methods for doing things that were more efficient than my own."
- "Different programming techniques for problems I had issues with"
- "By viewing other's programs it allowed me to think more critically about my own."
- "New ways of coding, different methods. Finally, how students read requirements differently."

Overall, eight of 20 student reviewers agreed that they learned from reviewing other students' work, and five of 20 disagreed. These findings suggest there is potential for peer review to enhance learning in computer programming classes.

# 6. CONCLUSION

This paper presents promising results concerning peer review for learning computer programming. We showed that peer reviews were accurate compared with a reasonable standard (a TA with industry experience). We also showed that students find reviews from other students with less experience to be most effective. Finally, we showed that reviewing other students' programming assignments increases overall learning outcomes. We also identified key issues for future

research, including developing incentive systems for eliciting honest reviews, algorithms for matching reviewers with authors, and review aggregation algorithms that are comprehensible to students. These findings illustrate the vast potential for distributed review in the classroom and beyond.

# 7. REFERENCES


[1] Anewalt, K. Using peer review as a vehicle for communication skill development and active learning. *J. Comput. Small Coll. 21*, 2 (2005), 148–155.

[2] Bloom, B., Englehart, M. D., Furst, E. J., Hill, W. H., and Krathwohl, D. R. *Taxonomy of Educational Objectives: The Classification of Educational Goals - Handbook 1: Cognitive Domain*. David McKay Company, Inc., New York, 1956.

[3] Cho, K., Chung, T. R., King, W. R., and Schunn, C. Peer-based computer-supported knowledge refinement: an empirical investigation. *Commun. ACM 51*, 3 (2008), 83–88.

[4] Clancey, W. J. From guidon to neomycin and heracles in twenty short lessons. *AI Mag. 7*, 3 (1986), 40–60.

[5] Collofello, J. S. Teaching technical reviews in a one-semester software engineering course. In *SIGCSE '87: Proceedings of the eighteenth SIGCSE technical symposium on Computer science education* (New York, NY, USA, 1987), ACM, pp. 222–227.

[6] Cosley, D., Frankowski, D., Terveen, L., and Riedl, J. Using intelligent task routing and contribution review to help communities build artifacts of lasting value. In *CHI '06: Proceedings of the SIGCHI conference on Human Factors in computing systems* (New York, NY, USA, 2006), ACM, pp. 1037–1046.

[7] Denning, T., Kelly, M., Lindquist, D., Malani, R., Griswold, W. G., and Simon, B. Lightweight preliminary peer review: does in-class peer review make sense? In *SIGCSE '07: Proceedings of the 38th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2007), ACM, pp. 266–270.

[8] Foltz, P. W., Laham, D., and Landauer, T. K. Automated essay scoring: Applications to education technology. In *Proceedings of ED-MEDIA* (1999), pp. 939–944.

[9] Gehringer, E. Strategies and mechanisms for electronic peer review. *Frontiers in Education Conference, 2000. FIE 2000. 30th Annual 1* (2000), F1B/2–F1B/7 vol.1.

[10] Gehringer, E. F. Electronic peer review and peer grading in computer-science courses. *SIGCSE Bull. 33*, 1 (2001), 139–143.

[11] Gehringer, E. F. Electronic peer review builds resources for teaching computer architecture. In *Proceedings of the 2003 American Society for Engineering Education Annual Conference & Exposition* (2003), American Society for Engineering Education.

[12] Gehringer, E. F., Chinn, D. D., Manuel A. Pérez-Qui n., and Ardis, M. A. Using peer review in teaching computing. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2005), ACM, pp. 321–322.

[13] Gehringer, E. F., Ehresman, L. M., and Skrien, D. J. Expertiza: students helping to write an ood text. In *OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (New York, NY, USA, 2006), ACM, pp. 901–906.

[14] Gotel, O., Scharff, C., and Wildenberg, A. Extending and contributing to an open source web-based system for the assessment of programming problems. In *PPPJ '07: Proceedings of the 5th international symposium on Principles and practice of programming in Java* (New York, NY, USA, 2007), ACM, pp. 3–12.

[15] Gotel, O., Scharff, C., and Wildenberg, A. Teaching software quality assurance by encouraging student contributions to an open source web-based system for the assessment of programming assignments. In *ITiCSE '08: Proceedings of the 13th annual conference on Innovation and technology in computer science education* (New York, NY, USA, 2008), ACM, pp. 214–218.

[16] Hamer, J., Kell, C., and Spence, F. Peer assessment using aropä. In *ACE '07: Proceedings of the ninth Australasian conference on Computing education* (Darlinghurst, Australia, Australia, 2007), Australian Computer Society, Inc., pp. 43–54.

[17] Hamer, J., Ma, K. T. K., and Kwong, H. H. F. A method of automatic grade calibration in peer assessment. In *ACE '05: Proceedings of the 7th Australasian conference on Computing education* (Darlinghurst, Australia, Australia, 2005), Australian Computer Society, Inc., pp. 67–72.

[18] Herlocker, J. L., Konstan, J. A., and Riedl, J. Explaining collaborative filtering recommendations. In *CSCW '00: Proceedings of the 2000 ACM conference on Computer supported cooperative work* (New York, NY, USA, 2000), ACM, pp. 241–250.

[19] Hinds, P. J. The curse of expertise: The effects of expertise and debiasing methods on predictions of novice performance. *Journal of Experimental Psychology: Applied 5*, 2 (1999), 205–221.

[20] Krathwohl, D. R., Bloom, B. S., and Masia, B. *Taxonomy of Educational Objectives: The Classification of Educational Goals - Handbook 2: Affective Domain*, 1 ed. Longman, London, UK, July 1964.

[21] Lampe, C., and Resnick, P. Slash(dot) and burn: distributed moderation in a large online conversation space. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems* (New York, NY, USA, 2004), ACM, pp. 543–550.

[22] Liu, E. Z.-F., Lin, S., Chiu, C.-H., and Yuan, S.-M. Web-based peer review: the learner as both adapter and reviewer. *Education, IEEE Transactions on 44*, 3 (Aug 2001), 246–251.

[23] Resnick, P., Kuwabara, K., Zeckhauser, R., and Friedman, E. Reputation systems. *Commun. ACM 43*, 12 (2000), 45–48.

[24] Silva, E., and Moreira, D. Webcom: a tool to use peer review to improve student interaction. *J. Educ. Resour. Comput. 3*, 1 (2003), 3.

[25] Sullivan, S. L. Reciprocal peer reviews. In *SIGCSE '94: Proceedings of the twenty-fifth SIGCSE symposium on Computer science education* (New York, NY, USA, 1994), ACM, pp. 314–318.

[26] Topping, K. Peer assessment between students in colleges and universities. *Review of Educational Research 68*, 3 (1998), 249–276.

[27] Trivedi, A., Kar, D. C., and Patterson-McNeill, H. Automatic assignment management and peer evaluation. *J. Comput. Small Coll. 18*, 4 (2003), 30–37.

[28] Trytten, D. Progressing from small group work to cooperative learning: a case study from computer science. *Frontiers in Education Conference, 1999. FIE '99. 29th Annual 2* (1999), 13A4/22–13A4/27 vol.2.

[29] Trytten, D. A. A design for team peer code review. In *SIGCSE '05: Proceedings of the 36th SIGCSE technical symposium on Computer science education* (New York, NY, USA, 2005), ACM, pp. 455–459.

[30] Wolfe, W. J. Online student peer reviews. In *CITC5 '04: Proceedings of the 5th conference on Information technology education* (New York, NY, USA, 2004), ACM, pp. 33–37.

[31] Zhang, J., Ackerman, M. S., and Adamic, L. Expertise networks in online communities: structure and algorithms. In *WWW '07: Proceedings of the 16th international conference on World Wide Web* (New York, NY, USA, 2007), ACM, pp. 221–230.