TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 07/11 pp435–445
DOI: 10.26599/TST.2018.9010108
Volume 24, Number 4, August 2019

Collaborative Assessments in Computer Science Education: A Survey

Hans Yuan and Paul Cao*

Abstract: As computer science enrollments continue to surge, assessments that involve student collaboration may play a more critical role in improving student learning. We provide a review on some of the most commonly adopted collaborative assessments in computer science, including pair programming, collaborative exams, and group projects. Existing research on these assessment formats is categorized and compared. We also discuss potential future research topics on the aforementioned collaborative assessment formats.

Key words: assessment; collaboration; pair programming; collaborative exam; group project

1 Introduction

An unprecedented number of students aim to study Computer Science (CS). Many universities report surging enrollment and the positive growth trend is continuing[1]. Computer science education faces many unique challenges. Firstly, computer science curriculum includes computer programming. Therefore, traditional pedagogies from other STEM (Science, Technology, Engineering, and Mathematics) areas may not be directly transferable. Secondly, the student body in computer science classes is becoming increasingly diverse, especially recently. A large number of non-majors participate in CS0 or CS1 classes. Thus, teaching pedagogies needs to cater to a diverse student body with varied technical background. Lastly, content in CS classes is not standardized. Courses with similar titles may vary dramatically across academic institutions on what is actually covered. This lack of standardization makes assessments of student learning difficult. Concept inventory and learning goals are still being developed[2]. These aforementioned challenges have attracted education researchers to explore and develop novel intervention and experiments to improve student learning in CS.

Assessments strongly influence student learning activities[3]. The so-called "hidden curriculum", i.e., student perceptions of what will appear in future assessments, dominates what and how long students study. Students usually try to achieve high study efficiency by aligning their study with the assessment.

Two types of assessments, summative and formative, have been extensively researched in the education literature. Summative assessments are usually high-stakes exams that focus on the assessment of what students know near the end of a study period (midterms and final exam). Formative assessments focus on using the results of the assessment to inform teachers how to improve student learning. Formative exams consist of low-stakes assessments such as in-class exercises and short quizzes[4]. Educators generally believe that students can achieve learning gains in formative assessment while also making learning gain in summative assessment through preparatory studies for the exam[4].

Assessments in computer science classes are varied and tend to include quizzes, exams, and programming assignments. Exams, one of the most widely used summative assessments, are used to gauge the effectiveness of instructions as well as student learning. Prior work[5] examined the benefits of testing under the theory of "testing memory" in laboratory studies and classroom experiments. They revealed that knowledge retention is improved by testing in at least the

• Hans Yuan and Paul Cao are with the Computer Science & Engineering Department, University of California, San Diego, La Jolla, CA 92093, USA. E-mail: {h3yuan, yic242}@eng.ucsd.edu.
* To whom correspondence should be addressed.
Manuscript received: 2018-06-01; accepted: 2018-06-26

short term[5]. The authors also argued for frequent testing which utilizes other mediated effects from testing. Low-stake formative assessments, such as in-class clicker questions and quizzes, provide frequent feedback to students. Most of the non-theory computer science classes also require students to complete programming assignments as a way to measure students' understanding of complicated algorithm and problem-solving skills.

The impact of peer collaborations on student learning has been studied in education research[6]. Significant progress was made within the CS education community for improving in-class student learning. They focused on in-class active learning pedagogies, such as Peer Instruction and POGIL (among others)[7,8]. However, recent work both outside computer science[9–16] and within computer science[17–20] has begun looking at examinations and programming exercises as additional opportunities to facilitate learning. Funding agencies, such as NSF, promote active learning and collaborations in STEM education[21].

Given the importance of assessments in CS education, especially those with collaborative aspects, we provide a review of current research on the selected assessment formats: pair programming, collaborative exams, and group projects. For each type of assessments, we review the state-of-the-art education research and provide our analysis of potential future research. The rest of the paper is organized as follows: For each section in Sections 2 – 4, we introduce the assessment, analyze and compare the existing studies, and provide potential research on the assessment format. We conclude the paper in Section 5.

2 Pair Programming

2.1 Introduction

One of the first uses of the term "pair programming" came from Cockburn and Williams[22] as they described pair programming's effectiveness in lowering financial and temporal cost, decreasing program defectiveness, reducing project management overhead, and fostering human relationships between programmers. The pair programming process involves two people at the same physical machine actively working together to develop a software solution. The pair consists of (1) the navigator, responsible for thinking about the high-level progression of the code, and (2) the driver, who controls the keyboard and mouse to type in the code. Together as a unit, the pair turns high-level ideas into code while thinking about how the desired program works. Periodically, the navigator and driver switch their roles.

2.2 Early research

Early research initially focused on pair programming's impact on student coding skill and examined the top contributing factors toward student success in introductory CS courses. Students' comfort level was the strongest predictor for success[23]. Over the years, use of pair programming in introductory CS resulted in strongly favorable feedback from students[24], higher retention rates[25], better quality homework code[26,27], and stronger confidence in students' own programming ability[28]. Students perceive their combined efforts result in stronger programs while increasing their confidence in their ability to do computer science.

Although pair programming benefits students, the effects are less pronounced when a pair has compatibility issues, which occur due to uneven skill level and possibly personality type[29,30].

The individuals' skill level, such as prior programming experience and perceived expertise, impacts the dynamics of the pair. For example, one of the partners may feel the burden of explaining prerequisite concepts to the partner[24]. The weaker student may feel intimidated and allow the stronger student to do all the work. Because of that, the dynamics of pair programming break down: the interaction between the driver and the navigator does not occur as desired.

A systematic review by Salleh et al.[30] found that some research concluded personality to be a significant factor, yet other research shows inconclusive results. The authors argued that the top indicators of incompatible pairs were (1) differences in actual skill level, (2) differences in perceived skill level, and (3) matching Myers-Briggs personality type. However, the authors found conflicting conclusions towards personality type as an indicator for compatibility. Most research studies saw no significant effect or mixed effect, while a couple showed a positive effect. None showed a negative effect. Because of these conclusions, Salleh et al.[30] argued that effort should be put into constructing pairs of similar skill level and possibly different personality types.

The effectiveness of pair programming was measured mainly in two ways: through student feedback and exam scores. Many of the early papers quantified

pair programming effectiveness by measuring project grades, exam grades, and questionnaires given to students[26, 27, 31]. They were then scored and averaged to convey how strongly students felt about a particular question. A few studies conducted interviews with students though many researchers considered test scores as students' real performance[24, 32, 33]. With pair programming as the uncertain variable, a few studies had the control group as no pair programming and the experimental group as some form of pair programming (shown in Table 1). Exams were then used to gauge performance before and after pair programming was used in the assignments.

2.3 Recent research

The cause of incompatibility was further investigated in recent research. Perceived and actual skill abilities impact the pair's performance because of one's lower or higher perception of the other[34]. A student who sees the partner as an expert would be affected by the perception, even if the other student's skill level is not at the expert level[32].

Based on the findings of earlier research suggesting the pairs be formed based on similar skill level, an attempt to do so measured students' performance in the class before pairing students together[39]. Students of similar ability perform just as well or better than a solo intro programmer with prior experience. The effects are more pronounced for lower percentile students, but this type of pairing does not hurt students' learning in the higher percentiles.

The success of pair programming hinges on the partners sharing a single computer and working on the same piece of code together. One study attempted to conduct *hybrid pair programming* where the pair is partners with separate computers with individual project code[45]. This dynamic breaks down the interactions between the partners and strengthens support for the unchanged version of pair programming.

Researchers have begun studying the long-term effects of pair programming in multiple settings. At the university, one study found that retention increased as a result of using pair programming in CS1 for female students and minorities, regardless of major[25]. Another study found that a combination "best practices" which incorporates pair programming, media computation, and peer instruction together into CS1, had lowered course drop rates and increased course pass rates for both men and women, majors and nonmajors[44].

Only very recently have researchers begun to determine whether all students did pair programming faithfully on homework projects. As they pair programmed, students were recorded[38, 45]. The data contained verbal and visual information not captured by scores and questionnaires, revealing phenomenon beyond performance. For example, Celepkolu and Boyer[45] codified levels of collaboration between the pair while Ruvalcaba et al.[38] observed specific differences in verbal communication between middle school students of different ethnic backgrounds.

Finally, researchers have begun to see how software design can improve learning software. Tutorial programs like PythonTutor[52] help people learn to program without physical interaction from another person. Novice programmers may experience frustrating setup overhead, and CodePilot[46] attempts to address this by having collaboration and version control built into the IDE. Another app acting as a tutor

Table 1 Pair programming research summary. This table summarizes research on pair programming based on the research topic and research methods.

		Pair compatibility				Demographics				Variations in design
		Perceived skill level	Actual skill level	Personality	Self-esteem	Nonmajor/ major	Gender	Ethnic group	Pre-university	
Research type	Observation	[24, 32, 34, 35]	[29]	[29, 34]	[29, 34, 36]		[36, 37]		[38]	
	experimentation	[39]	[39–42]	[25]	[43]	[31, 41, 44]	[25, 42, 44]	[25]		[33, 45, 46]
Type of data	Exams	[39, 40]				[31]	[36, 42]			
	Programming assignment	[39]	[42]			[31]			[38, 47–49]	[45, 50, 51]
	Questionnaire				[43]	[25]	[37]			[50]
	Peer evaluation	[29, 34]								
	Interviews	[24, 32]	[32]							[33]
	Retention rates					[25, 44]	[25, 44]	[25]		
	Recordings						[47]		[38, 47]	[33, 45, 51]
	Attendance					[41]	[37]			

measured the effect of pair programming and found that pairs required less help and spent less time than soloers[53].

In Table 1, we summarize recent research on pair programming and categorize based on their research topics and research methods. Research topics include compatibility, demographics, and variations in design. We divide research methods into observational studies and experiments. We also consider the type of data analyzed in each study.

2.4 Discussion

Based on the literature research, we think the following areas of pair programming might be promising in the future.

• **Effectiveness study.** Although many studies have measured pair programming's effectiveness, future studies may continue to improve the specific design and implementation of pair programming. One area of possible research is to continue measuring whether all pairs faithfully adhere to the rules of pair programming. Though assignments may be formatted for pair programming tasks, it was only occasionally guaranteed that the pairs programmed together rather than splitting the tasks among the pair to divide and conquer.

• **Apply pair programming to other domains.** Combinations of pair programming are used in various contexts, such as media computation and mobile programming. Pair programming helps undo a deterring perception that computer science is a field for the survival of the fittest[44]. In mobile programming, using the traditional pair model may not work identically since running an app may involve the use of another physical device[50]. Pairs may, instead of focusing on the same code on the same screen, focus on different screens and thus focus on different ideas.

• **Variations of pair programming.** Although pair programming is shown to be effective overall, there is still incentive to address issues faced by the few pairs not getting along. A pair of students may work in a lopsided manner: the stronger student completes most of the work while the weaker student misses important practice from the assignment. Pair programming models that counteract this type of phenomenon might be helpful. For example, two-stage programming assignments by Battestilli et al.[54] and its variations might be promising. It is also possible that software can help the pair programming model. Software interfaces can "tutor" students, or strengthen the pair programming process[53]. There is also the possibility that pairs use a software interface in which pair programming is the more natural way to succeed. Shared gaze may allow for remote pair programming to occur as effectively as in person. The fundamental principles that allow pair programming to succeed may become implemented in software, allowing people to focus on the same code at the same time[33].

• **Pairing policy.** Alternative policies to pair by ability can be explored, such as mental models. Although some studies have framed pair programming as a cognitive phenomenon (distributed cognition, mental model)[22,55], researchers examined the use of mental model consistency as a pairing policy for assignments[56]. Furthermore, to the best of our knowledge, no work exists to measure the relationship between prior experience and mental model consistency.

3 Collaborative Exam

3.1 Introduction

In addition to guiding students what and how to study for a certain course, both formative and summative assessments also provide valuable feedback to students on their performance. Existing research has shown that quality feedback may have one of the most influential effects in student learning[57,58]. In the context of formative assessment, the role of feedback is crucial. According to Gibbs et al.[59,60], the effectiveness of feedback can be improved by following the guidelines below.

- Feedback should be sufficiently detailed and often enough.
- Feedback should be focused on students' performance and learning with actions that are under students' control.
- Feedback should be timely, preferably before students know their marks on the assessment.
- Feedback should be utilized by students.

In computer science education, the effects of feedback have been studied. For example, Epstein et al.[61] used three experiments to investigate the effects of immediate feedback in assessments and found the technique leads to better knowledge retention after a week. Stone[62] studied the effect of reflective blogs as a way to provide frequent feedback to students. Results of this study show that these types of feedback have affected students' confidence,

frustration, and engagement, mostly in a positive manner. However, assessment feedback can be viewed differently depending on the timing of feedback. If feedback is provided together with the marks of an assessment, students tend to over-focus on the exam scores instead of examining and reflecting on the feedback[4]. Another realistic constraint is the time it takes for instructors to provide meaningful and constructive feedback. The time constraint can be remedied by letting students discuss through peer instruction.

To make summative exams a learning experience for students, collaborative exams can be used as a formative assessment where students work cooperatively[63, 64]. A variant of collaborative exams is two-stage exams[65]. Two-stage exams start with a normal individual exam immediately followed by a group exam where students work in small groups to solve a set of problems. The group exam component provides instant feedback to students as they work toward answering the questions in a high-stakes exam environment. The two-stage exam format can be viewed both as a formative assessment (the collaborative portion) and a summative assessment (the individual portion). Students receive instant feedback from their peers on the content before they know their grades. Due to the high-stakes nature of the exam, students are expected to actively contribute to finding the best solution in the collaborative portion.

Educators in different fields have embraced this testing strategy. In geology and oceanography, quasi-controlled experiments have shown that collaborative testing aids poorly performing students[13, 15]. Another study in geology used historical data to show that collaborative testing improves attendance and student grades[66]. In mechanical engineering, collaborative testing has been similarly shown to improve student grades[67]. In physics, the reason why students value the collaborative testing format has been explored, and found students have overwhelmingly positive opinions towards two-stage exams[14]. Students reported that the exam format tends to be enjoyable and helpful for learning. It may also increase confidence and provide immediate feedback. To find a better grouping strategy, team dynamics during two-stage exams has also been explored[68].

3.2 Existing research

Two-stage exams have been examined for their effectiveness in promoting student learning using controlled experiments in non-computing disciplines. Using a pre-post test design, Cortright et al.[9] used a crossover design to explore short-term learning benefits from group exams, based on performance on a follow-up exam given four weeks later. They found the group exam improved student knowledge retention. A similar crossover design was used in physics to find that student improvement from two-stage exams is significant within 1–2 weeks but vanishes after 6–7 weeks[12]. A potential confounding factor to these studies was that more time was spent only on re-tested topics during the group exam. Addressing this potential problem, Gilley and Clarkston[11] added an individual retest to the crossover design. The individual retest mirrors the group test in time and allows for differentiating learning between simply having more time on a topic and having more time in the context of a collaborative exam. Their work concluded that the group component of the two-stage exam improves student learning in geology.

In computer science, Yu et al.[18] reported on the positive attitudes from students on the testing format. This study also examined student learning through the use of isomorphic questions on midterms and final exams to measure two-stage exam effectiveness, but the results were inconclusive[18]. Cao and Porter[19] completed the first quantitative study of two-stage exams in computer science which showed significantly improved student learning through a pre-post test experiment in CS1.5[19]. Two sections of CS1.5 (Java) in the Spring of 2016 were selected for this study as CS1.5 is a gateway course into the major. Peer instruction[8] is used in this course. For each midterm, two CS topics were pre-selected. The individual exam (40 minutes) includes one problem for each of the pre-selected topics in addition to other normal topics covered in the previous two weeks. Individual exam papers were collected before the start of the "individual-retest" phase (12 minutes). The class was divided into halves with each half answering a single problem on one of the two pre-selected topics. The individual-retest phase tries to compensate for the time differences that students spent on the two pre-selected topics. In the end, the group-retest phase (or simply called the group phase) begins with four students in randomly pre-assigned groups[18] with two to three groups with five students. Each group answers a question on the topic that the other half of the class answered during the individual-retest phase. The group test phase lasts about

18 minutes depending on the time it took to collect and distribute papers. The entire test period is 80 minutes. Two weeks after the midterm, a scheduled quiz (30 minutes) was given which included problems on topics covered in the past two weeks and one problem on each of the topics tested in the midterm.

The overall effectiveness of the group exam is determined by comparing normalized gains for students exposed to the control (individual-retest) versus the experimental condition (group-retest). Because of the randomized crossover exam, each student is in the control group for one topic and in the experimental group for the other topic. Paired t-test on student normalized learning gain was used to determine statistical significance. For the first experiment, students gained both from the group-retest and individual-retest, but those in the group-retest gained more. For the second experiment, students performed worse on the post-test and hence experienced negative learning gain. Recall that the questions were not designed to be isomorphic, so the loss in performance could be due to differences in question difficulty. Students in the group-retest performed better by experiencing a smaller reduction in normalized learning gain. Both experiments have small to moderate effect sizes. During the initial study of two-stage exams in computer science[19], data on how students view their exam papers were collected using an online grading platform. The results showed that there is no difference between the control and experimental groups in the percentage of students who either view their exam papers or not. Therefore, two-stage exams do not seem to improve the rate of exam pickup from students though they improve student learning in the short term. We believe that students developing better exam preparation strategies as well as evaluating their performances are important. Hence, we propose to use exam wrappers as a tool to increase students' metacognitive skills and improve their understanding of course contents.

In a follow-up study, Cao and Porter[20] investigated the first study in computer science on how collaborative exams affect student learning based on students' performance level, both individually and collaboratively in a group. They found that mid-performing students show significant improvement in their post-test compared with their per-test after collaborative exams while low- and high-performing students tend to have insignificant performance differences. Using group members' performance levels in the pre-test to measure group heterogeneity, we find that low and moderate heterogeneity groups statistically benefit from collaborative exams. These results suggest the homogeneous grouping of students by performance when employing group exams.

3.3 Discussion

As the effectiveness of collaborative exams has been identified, more research on variations and composition of group members are still needed. Based on the literature research, the following research areas could be potentially explored.

What kind of questions are the most effective in the collaborative phase of the exam? To study the effect of collaborative exams immediately following a normal individual summative midterm, two possible types of questions can be given in the collaborative phase. Firstly, selected identical questions from the individual exam can be given again to groups of students. Or, students may be given questions that are different from but related to those in the individual exam during the collaborative phase. Existing literature on two-stage exams primarily focuses on identical or isomorphic questions for effective studies[11], but there is evidence that related questions which are more involved will have better effects[19].

What is the optimal team composition during the collaborative phase to achieve better student learning, especially for underrepresented students in computer science such as female and minorities? Computer science is undergoing an enrollment boom recently, but the percentage of female students and minorities in this area is still low[69–71]. Thus interventions that may improve unrepresented students' learning have been an active research area. In the context of collaborative exams, group tests have been shown to improve student performance on subsequent questions with low-performing students benefiting more[10]. However, no existing work has studied the effect of grouping strategies on underrepresented students.

To what extent does the timing of the collaborative part of the exam affect student learning? To gain the full benefit of this exam format, 80-minute or longer exam time is necessary as students need sufficient time to discuss and find a solution. As most of the midterm exams are held in class, collaborative exams have rarely been applied to 50-minute classes. To remedy

the time constraints of a 50-minute class, we propose to investigate if a delayed collaborative exam held days after a normal individual exam also has similar benefits to improve student learning. The impact of this research result may significantly improve the adaptation of collaborative exams in undergraduate classes.

4 Group Project

We want to begin answering the question *"how has collaborative assessment been done within group project courses such as SoftWare Engineering (SWE) and Human-Computer Interaction (HCI)?"* Although there is a plethora of research on the courses themselves, they focus more on designing the course than evaluating the effect of the course on student learning.

Furthermore, what is defined as "assessment" in these courses? Unlike CS1 material, which can be tested directly via written exams and programming assignments, SWE and HCI group project success is not so easily defined due to their subjective nature. The most common metric for the following case studies is student satisfaction with the course.

For SWE courses, ways to measure success include the number of lines of code in a project, the maintainability of the object-oriented design, and software companies' interest in students' work in the projects[72]. For HCI courses, the usability of high-level design and the apparent progress rather than the final product may indicate student performance[73]. Despite the subjectivity of these measurements, students feel they learned a substantial amount when working the SWE/HCI group projects that felt realistic. Attempts to improve project courses towards realistic environments can be categorized into two groups:

- Integrating the "real world" into projects;
- Introducing students to "real world" projects.

In this section, we focus on a small subset of research studies that focuses on how making projects realistic affects the efficacy of SWE and HCI courses for student group learning.

4.1 Making academic projects realistic

A natural first answer to preparing student groups for the "real world" setting is to manipulate how groups succeed in the academic setting. One way to change groupings is to have every enrolled student (both undergraduate and graduate) in a course work together on the same massive project[74]. Due to the scale, no individual student can know every detail of the project. Students embody different roles such as team lead, project manager, and developer. They use tools such as online discussion boards and issue trackers to maintain communication and progress. The study found from the student feedback that (1) while SWE or project manager experience was useful, there was too much management overhead and (2) the lack of a "domain expert" detracted from the realistic environment of the project. Since academic projects are constrained to time frames set by the school and students may have differing obligations, the detailed replication of an SWE environment may have unintended conflicts with the temporal resources students have. Also, students' goal in the academic setting is to learn and thus have different motivation from that of a software engineer in a company.

Unlike pair programming and the aforementioned student project, which consists of theoretically equally-contributing team members aspiring for the same goal, groups can consist of people with varying roles. One way is to include non-student mentors for the group members. For example, Hartfield et al.[73] invited experts from the industry to act as mentors for the HCI class student groups. Since the mentors do not affect students' grades, the students can honestly describe design shortcomings to solicit candid advice and suggestions. The assessment is formative because students can improve their work without worrying about their grades. The author reported that students learned a "tremendous amount" and that this course ranked twice in the 99th percentile in the School of Engineering at Stanford University for "overall value of [a] course".

These two ideas had one thing in common: the group composition was heterogeneous due to the inclusion of professionals[73] and graduate students with prior industry experience[74]. Furthermore, the groups have different roles (lead, developer, and mentor). Because of the heterogeneous groups, project scale, and logistical challenges, the groups worked in a more realistic environment. Student feedback is more positive as they appreciate working on projects with grounded aspirations.

4.2 Projects in the real world

In the "real world", people live around the globe, speak different languages, and have different needs/problems. The industry is no exception to this rule — worldwide collaboration must work with time zones and language

differences. To prepare students for a globalized world, Petkovic et al.[72] designed an SWE course where some student groups consist of students from San Francisco, California and others from Fulda, Germany. The practical obstacles of the SWE course such as scheduling and communicating are amplified by time zone and language differences. Despite the additional challenges, both local-only groups and global groups overcame their challenges just as well. Student feedback for the course was strongly positive, the project attracted job interviewers' interest, and the lessons from the class directly translated to students' work environments. Since the main challenges in SWE are to coordinate efforts between people (colloquially termed "soft skills"), the main focus of SWE courses may not only be attaining programming ability. An internationally coordinated project signals to job interviewers the possible experience the student gained from overcoming obstacles like time zones. Because students found the project applicable to "real world" scenarios, we believe that the project reached a level of realistic above traditional group project classes.

Another consideration of group projects is the problem the product tries to solve — how realistic is the problem? To implement a real product, it must have a legitimate purpose to address people's needs. Thus, students would need to talk to real clients and users. The benefit of this approach is that students work on a real project[75]. They worked with a Dutch banking company whose users needed better ATM interfaces for illiterate or foreign populations. To talk designs with users, the students visited a school that taught introductory Dutch and tested their designs with a population that closely resembles their target users. The students' projects had a realistic aspect that would not be present in the traditional group project class. In their open-response survey question, the authors found that the students thought the involvement of clients and users were important and that the project was interesting. When working on a project based on a real need, the context of the project is not contrived or invented. Because students worked on a project whose premise is realistic, this structure of group project proved more motivating and instructional for students.

While software implementations and engineering remain important, the question is becoming important, *"What can computers do for us?"* The sample of group project classes seen thus far showed how an instructor might introduce realistic elements into the class or introduce the student to the real world. However, it is also possible that the needs of the real world drive the course itself[76]. In addition to learning design, managing tasks, communicating within a team, and iterating prototypes, students also "learned compassion". When assessing students, the focus has been on technical knowledge and skill. However, as technology becomes more involved in people's lives, empathizing to design and engineer solutions that is usable by other people may become a type of assessment worth considering.

5 Conclusion

Collaboration among students during formative and summative assessments may significantly improve student learning. We provide a systematic review on pair programming, collaborative exam, and group projects. Existing research has shown that students exposed to pair programming and collaborative exams performed more strongly. For advanced students learning SWE and HCI in groups, the direct pedagogical effect of the course on learning was measured on the quality of their work and the satisfaction derived from doing their projects. This paper proposed that future research on pair programming might focus on continued effectiveness studies, the application to other domain, and variations of pair programming and pairing policies. For collaborative exam, the type of questions, team compositions, and timing of the collaborative phase may be examined.

References

[1] Computing Research Association, The Taulbee survey, http://cra.org/crn/wp-content/uploads/sites/7/2017/05/2016-Taulbee-Survey.pdf, August 25, 2017.

[2] L. Porter, D. Zingaro, C. Lee, C. Taylor, K. C. Webb, and M. Clancy, Developing course-level learning goals for basic data structures in CS2, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 858–863.

[3] B. R. Snyder, *The Hidden Curriculum*. New York, NY, USA: Knopf, 1970.

[4] R. E. Bennett, Formative assessment: A critical review, *Assessment in Education: Principles, Policy & Practice*, vol. 18, no. 1, pp. 5–25, 2011.

[5] H. L. Roediger and J. D. Karpicke, The power of testing memory: Basic research and implications for educational practice, *Perspectives on Psychological Science*, vol. 1, no. 3, pp. 181–210, 2006.

[6] S. Freeman, S. L. Eddy, M. McDonough, M. K. Smith, N. Okoroafor, H. Jordt, and M. P. Wenderoth, Active learning

increases student performance in science, engineering, and mathematics, *Proc. Natl. Acad. Sci. USA*, vol. 111, no. 23, pp. 8410–8415, 2014.

[7] H. H. Hu, C. Kussmaul, B. Knaeble, C. Mayfield, and A. Yadav, Results from a survey of faculty adoption of process oriented guided inquiry learning (POGIL) in computer science, in *Proc. 2016 ACM Conf. Innovation and Technology in Computer Science Education*, Arequipa, Peru, 2016, pp. 186–191.

[8] L. Porter, C. B. Lee, B. Simon, and D. Zingaro, Peer instruction: Do students really learn from peer discussion in computing? in *Proc. 7^{th} Int. Workshop on Computing Education Research*, Providence, RI, USA, 2011, pp. 45–52.

[9] R. N. Cortright, H. L. Collins, D. W. Rodenbaugh, and S. E. DiCarlo, Student retention of course content is improved by collaborative-group testing, *Advances in Physiology Education*, vol. 27, no. 3, pp. 102–108, 2003.

[10] Ö. Dahlström, Learning during a collaborative final exam, *Educational Research and Evaluation*, vol. 18, no. 4, pp. 321–332, 2012.

[11] B. H. Gilley and B. Clarkston, Collaborative testing: Evidence of learning in a controlled in-class study of undergraduate students, *Journal of College Science Teaching*, vol. 43, no. 3, pp. 83–91, 2014.

[12] J. Ives, Measuring the learning from two-stage collaborative group exams, arXiv preprint arXiv: 1407.6442, 2014.

[13] G. L. Macpherson, Y. J. Lee, and D. Steeples, Group-examination improves learning for low-achieving students, *Journal of Geoscience Education*, vol. 59, no. 1, pp. 41–45, 2011.

[14] G. W. Rieger and C. E. Heiner, Examinations that support collaborative learning: The students' perspective, *Journal of College Science Teaching*, vol. 43, no. 4, pp. 41–47, 2014.

[15] R. F. Yuretich, S. A. Khan, R. M. Leckie, and J. J. Clement, Active-learning methods to improve student performance and scientific interest in a large introductory oceanography course, *Journal of Geoscience Education*, vol. 49, no. 2, pp. 111–119, 2001.

[16] J. F. Zipp, Learning by exams: The impact of two-stage cooperative tests, *Teaching Sociology*, vol. 35, no. 1, pp. 62–76, 2007.

[17] M. Craig, D. Horton, D. Zingaro, and D. Heap, Introducing and evaluating exam wrappers in CS2, in *Proc. 47^{th} ACM Technical Symp. Computing Science Education*, Memphis, TN, USA, 2016, pp. 285–290.

[18] B. Yu, G. Tsiknis, and M. Allen, Turning exams into a learning experience, in *Proc. 41^{st} ACM Technical Symp. Computer Science Education*, Milwaukee, WI, USA, 2010, pp. 336–340.

[19] Y. J. Cao and L. Porter, Evaluating student learning from collaborative group tests in introductory computing, in *Proc. 2017 ACM SIGCSE Technical Symp. Computer Science Education*, Seattle, WA, USA, 2017, pp. 99–104.

[20] Y. J. Cao and L. Porter, Impact of performance level and group composition on student learning during collaborative exams, in *Proc. 2017 ACM Conf. Innovation and Technology in Computer Science Education*, Bologna, Italy, 2017, pp. 152–157.

[21] National Science Foundation, Improving Undergraduate STEM Education: Education and Human Resources (IUSE: EHR), https:// www.nsf.gov/pubs/2017/nsf17590/ nsf17590.htm, 2018.

[22] A. Cockburn and L. Williams, The costs and benefits of pair programming, in *Extreme Programming Examined*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001, pp. 223–243.

[23] B. C. Wilson and S. Shrock, Contributing to success in an introductory computer science course: A study of twelve factors, in *Proc. 32^{nd} SIGCSE Technical Symp. Computer Science Education*, Charlotte, NC, USA, 2001, pp. 184–188.

[24] B. Simon and B. Hanks, First-year students' impressions of pair programming in CS1, *J. Educ. Resour. Comput.*, vol. 7, no. 4, p. 5, 2008.

[25] J. C. Carver, L. Henderson, L. L. He, J. Hodges, and D. Reese, Increased retention of early computer science and software engineering students using pair programming, in *Proc. 20^{th} Conf. Software Engineering Education & Training*, Dublin, Ireland, 2007, pp. 115–122.

[26] C. McDowell, L. Werner, H. Bullock, and J. Fernald, The effects of pair-programming on performance in an introductory programming course, in *Proc. 33^{rd} SIGCSE Technical Symp. Computer Science Education*, Cincinnati, KY, USA, 2002, pp. 38–42.

[27] C. McDowell, B. Hanks, and L. Werner, Experimenting with pair programming in the classroom, in *Proc. 8^{th} Annual Conf. Innovation and Technology in Computer Science Education*, Thessaloniki, Greece, 2003, pp. 60–64.

[28] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald, Pair programming improves student retention, confidence, and program quality, *Communications of the ACM*, vol. 49, no. 8, pp. 90–95, 2006.

[29] N. Katira, L. Williams, E. Wiebe, C. Miller, S. Balik, and E. Gehringer, On understanding compatibility of student pair programmers, in *Proc. 35^{th} SIGCSE Technical Symp. Computer Science Education*, Norfolk, VA, USA, 2004, pp. 7–11.

[30] N. Salleh, E. Mendes, and J. Grundy, Empirical studies of pair programming for CS/SE teaching in higher education: A systematic literature review, *IEEE Trans. Softw. Eng.*, vol. 37, no. 4, pp. 509–525, 2011.

[31] N. Nagappan, L. Williams, M. Ferzli, E. Wiebe, K. Yang, C. Miller, and S. Balik, Improving the CS1 experience with pair programming, *ACM SIGCSE Bull.*, vol. 35, no. 1, pp. 359–362, 2003.

[32] A. Tafliovich, J. Campbell, and A. Petersen, A student perspective on prior experience in CS1, in *Proc. 44^{th} ACM Technical Symp. Computer Science Education*, Denver, CO, USA, 2013, pp. 239–244.

[33] S. D'Angelo and A. Begel, Improving communication between pair programmers using shared gaze awareness, in *Proc. 2017 CHI Conf. Human Factors in Computing Systems*, Denver, CO, USA, 2017, pp. 6245–6290.

[34] J. Sennett and M. Sherriff, Compatibility of partnered students in computer science education, in *Proc. 41st ACM Technical Symp. Computer Science Education*, Milwaukee, WI, USA, 2010, pp. 244–248.

[35] M. Celepkolu and K. E. Boyer, Thematic analysis of students' reflections on pair programming in CS1, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 771–776.

[36] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald, The impact of pair programming on student performance, perception and persistence, in *Proc. 25th Int. Conf. Software Engineering*, Portland, OR, USA, 2003, pp. 602–607.

[37] O. Aarne, P. Peltola, J. Leinonen, and A. Hellas, A study of pair programming enjoyment and attendance using study motivation and strategy metrics, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 759–764.

[38] O. Ruvalcaba, L. Werner, and J. Denner, Observations of pair programming: Variations in collaboration across demographic groups, in *Proc. 47th ACM Technical Symp. Computing Science Education*, Memphis, TN, USA, 2016, pp. 90–95.

[39] G. Braught, J. MacCormick, and T. Wahls, The benefits of pairing by ability, in *Proc. 41st ACM Technical Symp. Computer Science Education*, Milwaukee, WI, USA, 2010, pp. 249–253.

[40] G. Braught, L. M. Eby, and T. Wahls, The effects of pair-programming on individual programming skill, in *Proc. 39th SIGCSE Technical Symp. Computer Science Education*, Portland, OR, USA, 2008, pp. 200–204.

[41] C. O'Donnell, J. Buckley, A. Mahdi, J. Nelson, and M. English, Evaluating pair-programming for non-computer science major students, in *Proc. 46th ACM Technical Symp. Computer Science Education*, Kansas City, MO, USA, 2015, pp. 569–574.

[42] G. Braught, T. Wahls, and L. M. Eby, The case for pair programming in the computer science classroom, *Trans. Comput. Educ.*, vol. 11, no. 1, p. 2, 2011.

[43] M. M. Muller and F. Padberg, An empirical study about the feelgood factor in pair programming, in *Proc. 10th Int. Symp. Software Metrics*, Chicago, IL, USA, 2004, pp. 151–158.

[44] L. Porter and B. Simon, Retaining nearly one-third more majors with a trio of instructional best practices in CS1, in *Proc. 44th ACM Technical Symp. Computer Science Education*, Denver, CO, USA, 2013, pp. 165–170.

[45] M. Celepkolu and K. E. Boyer, The importance of producing shared code through pair programming, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 765–770.

[46] J. Warner and P. J. Guo, CodePilot: Scaffolding end-to-end collaborative software development for novice programmers, in *Proc. 2017 CHI Conf. Human Factors in Computing Systems*, Denver, CO, USA, 2017, pp. 1136–1141.

[47] J. Tsan, F. J. Rodríguez, K. E. Boyer, and C. Lynch, "I think we should... ": Analyzing elementary students' collaborative processes for giving and taking suggestions, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 622–627.

[48] L. Werner, J. Denner, S. Campe, E. Ortiz, D. DeLay, A. C. Hartl, and B. Laursen, Pair programming for middle school students: Does friendship influence academic outcomes? in *Proc. 44th ACM Technical Symp. Computer Science Education*, Denver, CO, USA, 2013, pp. 421–426.

[49] C. M. Lewis and N. Shah, How equity and inequity can emerge in pair programming, in *Proc. 11th Annu. Int. Conf. Inter. Computing Education Research*, Omaha, NE, USA, 2015, pp. 41–50.

[50] M. Seyam and D. S. McCrickard, Teaching mobile development with pair programming, in *Proc. 47th ACM Technical Symp. Computing Science Education*, Memphis, TN, USA, 2016, pp. 96–101.

[51] F. J. Rodríguez, K. M. Price, and K. E. Boyer, Exploring the pair programming process: Characteristics of effective collaboration, in *Proc. 2017 ACM SIGCSE Technical Symp. Computer Science Education*, Seattle, WA, USA, 2017, pp. 507–512.

[52] P. J. Guo, Online python tutor: Embeddable web-based program visualization for CS education, in *Proc. 44th ACM Technical Symp. Computer Science Education*, Denver, CO, USA, 2013, pp. 579–584.

[53] R. Harsley, D. Fossati, B. Di Eugenio, and N. Green, Interactions of individual and pair programmers with an intelligent tutoring system for computer science, in *Proc. 2017 ACM SIGCSE Technical Symp. Computer Science Education*, Seattle, WA, USA, 2017, pp. 285–290.

[54] L. Battestilli, A. Awasthi, and Y. J. Cao, Two-stage programming projects: Individual work followed by peer collaboration, in *Proc. 49th ACM Technical Symp. Computer Science Education*, Baltimore, ML, USA, 2018, pp. 479–484.

[55] A. Radermacher, G. Walia, and R. Rummelt, Improving student learning outcomes with pair programming, in *Proc. 9th Annu. Int. Conf. Int. Computing Education Research*, Auckland, New Zealand, 2012, pp. 87–92.

[56] A. Radermacher, G. Walia, and R. Rummelt, Assigning student programming pairs based on their mental model consistency: An initial investigation, in *Proc. 43rd ACM Technical Symp. Computer Science Education*, Raleigh, NC, USA, 2012, pp. 325–330.

[57] B. J. Fraser, H. J. Walberg, W. W. Welch, and J. A. Hattie, Syntheses of educational productivity research, *International Journal of Educational Research*, vol. 11, no. 2, pp. 147–252, 1987.

[58] P. Black, Formative assessment: Raising standards inside the classroom, *School Science Review*, vol. 80, no. 291, pp. 39–46, 1998.

[59] G. Gibbs and C. Simpson, Conditions under which assessment supports students' learning, *Learning and Teaching in Higher Education*, vol. 1, no. 1, pp. 3–31, 2004.

[60] A. Cain and M. Ali Babar, Reflections on applying constructive alignment with formative feedback for teaching introductory programming and software architecture, in *2016 IEEE/ACM 38th Int. Conf. Software Engineering Companion*, Austin, TX, USA, 2016, pp. 336–345.

[61] M. L. Epstein, A. D. Lazarus, T. B. Calvano, K. A. Matthews, R. A. Hendel, B. B. Epstein, and G. M. Brosvic, Immediate feedback assessment technique promotes learning and corrects inaccurate first responses, *The Psychological Record*, vol. 52, no. 2, pp. 187–201, 2002.

[62] J. A. Stone, Using reflective blogs for pedagogical feedback in CS1, in *Proc. 43rd ACM Technical Symp. Computer Science Education*, Raleigh, NC, USA, 2012, pp. 259–264.

[63] P. Heller, R. Keith, and S. Anderson, Teaching problem solving through cooperative grouping, Part 1: Group versus individual problem solving, *American Journal of Physics*, vol. 60, no. 7, pp. 627–636, 1992.

[64] S. A. Stearns, Collaborative exams as learning tools, *College Teaching*, vol. 44, no. 3, pp. 111–112, 1996.

[65] D. Cohen and J. Henle, The pyramid exam, *UME Trends*, vol. 10, no. 2, pp. 15–16, 1995.

[66] K. Knierim, H. Turner, and R. K. Davis, Two-stage exams improve student learning in an introductory geology course: Logistics, attendance, and grades, *Journal of Geoscience Education*, vol. 63, no. 2, pp. 157–164, 2015.

[67] M. Fengler and P. M. Ostafichuk, Successes with two-stage exams in mechanical engineering, in *Proc. Canadian Engineering Education Association (CEEA) Conf.*, 2015, pp. 1–5.

[68] I. D. Beatty, Collaboration or copying? Student behavior during two-phase exams with individual and team phases, in *Physics Education Research Conf. 2015*, College Park, MD, USA, 2015, pp. 59–62.

[69] L. Barker, C. L. Hovey, and L. D. Thompson, Results of a large-scale, multi-institutional study of undergraduate retention in computing, in *2014 IEEE Frontiers in Education Conf. (FIE) Proc.*, Madrid, Spain, 2014, pp. 1–8.

[70] S. Beyer, M. DeKeuster, K. Walter, M. Colar, and C. Holcomb, Changes in CS students' attitudes towards CS over time: An examination of gender differences, *ACM SIGCSE Bull.*, vol. 37, no. 1, pp. 392–396, 2005.

[71] C. Alvarado, Y. J. Cao, and M. Minnes, Gender differences in students' behaviors in CS classes throughout the CS major, in *Proc. 2017 ACM SIGCSE Technical Symp. Computer Science Education*, Seattle, WA, USA, 2017, pp. 27–32.

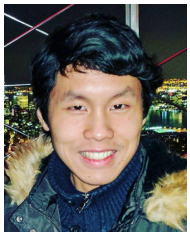
[72] D. Petkovic, G. Thompson, and R. Todtenhoefer, Teaching practical software engineering and global software engineering: Evaluation and comparison, in *Proc. 11th Annual SIGCSE Conf. Innovation and Technology in Computer Science Education*, Bologna, Italy, 2006, pp. 294–298.

[73] B. Hartfield, T. Winograd, and J. Bennett, Learning HCI design: Mentoring project groups in a course on human-computer interaction, in *Proc. 23rd SIGCSE Technical Symp. Computer Science Education*, Kansas City, MO, USA, 1992, pp. 246–251.

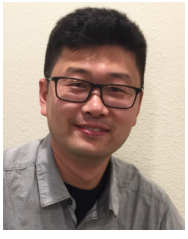
[74] D. Coppit and J. M. Haddox-Schatz, Large team projects in software engineering courses, in *Proc. 36th SIGCSE Technical Symp. Computer Science Education*, St. Louis, MO, USA, 2005, pp. 137–141.

[75] H. Koppelman and B. van Dijk, Creating a realistic context for team projects in HCI, in *Proc. 11th Annual SIGCSE Conf. Innovation and Technology in Computer Science Education*, Bologna, Italy, 2006, pp. 58–62.

[76] M. Bratton, Global TIES: Ten years of engineering for humanity, *International Journal for Service Learning in Engineering*, vol. 9, pp. 205–221, 2014.



Hans Yuan received the BS degree from University of California, San Diego, USA in 2018. He developed course material for introductory computer science courses, studied online tech-enhanced learning, and did research with an ambulance dispatch software project. He is interested in education efficacy and humanitarian design and engineering. Now, he is a Cloud Software Engineer at Survey Monkey by day and continues ambulance dispatch research by night.



Paul Cao received the BS degree in engineering from Nanjing University of Science & Technology in 2001, and PhD degree from Duke University (USA) in 2006. He taught at Oberlin College and Ashland University and is currently a lecturer in the Department of Computer Science & Engineering at University of California, San Diego, USA. He is a member of ACM and IEEE. His research interest is computer science education and mobile data analysis.