Designing for learning during collaborative projects online: tools and takeaways

Sreecharan Sankaranarayanan, Siddharth Reddy Kandimalla, Mengxin Cao, Ignacio Maronna, Haokang An, Chris Bogart, R. Charles Murray, Michael Hilton, Majd Sakr and Carolyn Penstein Rosé
(Author affiliations can be found at the end of the article)

Received 11 April 2020
Revised 8 May 2020
25 May 2020
2 June 2020
Accepted 3 June 2020

Abstract

Purpose – In response to the evolving COVID-19 pandemic, many universities have transitioned to online instruction. With learning promising to be online, at least in part, for the near future, instructors may be thinking of providing online collaborative learning opportunities to their students who are increasingly isolated from their peers because of social distancing guidelines. This paper aims to provide design recommendations for online collaborative project-based learning exercises based on this research in a software engineering course at the university level.

Design/methodology/approach – Through joint work between learning scientists, course instructors and software engineering practitioners, instructional design best practices of alignment between the context of the learners, the learning objectives, the task and the assessment are actualized in the design of collaborative programming projects for supporting learning. The design, first segments a short real-time collaborative exercise into tasks, each with a problem-solving phase where students participate in collaborative programming, and a reflection phase for reflecting on what they learned in the task. Within these phases, a role-assignment paradigm scaffolds collaboration by assigning groups of four students to four complementary roles that rotate after each task.

Findings – By aligning each task with granular learning objectives, significant pre- to post-test learning from the exercise as well as each task is observed.

Originality/value – The roles used in the paradigm discourage divide-and-conquer tendencies often associated with collaborative projects. By requiring students to discuss conflicting ideas to arrive at a consensus implementation, their ideas are made explicit, thus providing opportunities for clarifying misconceptions through discussion and learning from the collaboration.

Keywords Instructional design, Higher education, Software engineering, Computer-supported collaborative learning, Mob programming, Online collaborative Programming, Project-based online course

Paper type Research paper

This work was funded in part by NSF grants IIS 1822831, IIS 1917955, and funding from Microsoft.

This article is part of the special issue, "A Response to Emergency Transitions to Remote Online Education in K-12 and Higher Education," which contains shorter, rapid-turnaround invited works, not subject to double blind peer review. The issue was called, managed and produced on short timeline in Summer 2020 toward pragmatic instructional application in the Fall 2020 semester.

Information and Learning Sciences

2398-5348
DOI 10.1108/ILS-04-2020-0095

Background and motivation

Collaborative projects in project-based courses can be opportunities for students to observe, engage with and learn from other students who might embody different styles of work, bring different ideas to bear on a given project, and demonstrate the skills required to execute a given idea. They are also designed to be more complex than individual tasks and as a result, can allow students to benefit from collaborating with group members rather than tackling the task alone. In the higher education context, when designed as authentic tasks, they can allow students to experience what collaboration would be like in their eventual workplace teams. Learning from these collaborative projects, however, often conflicts with the tendency of students to focus on the grade obtained for completion. Consequently, in a race to maximize efficiency, the collaboration can devolve into a "divide-and-conquer" approach with each student focusing on the part of a project they are already good at, missing out on learning from the collaboration as a result. The challenges for collaborating on projects become even more pronounced online as differing student schedules make it harder to work together on projects. This results in students defaulting to the "path of least resistance" and choosing parts of the project to complete on their own time.

To dissuade divide-and-conquer while creating authentic learning experiences for individuals participating in online collaborative projects, we present in this paper, a paradigm, called the online programming exercise (OPE), based on theoretical ideas from instructional design and the learning sciences. The design of the OPE paradigm is investigated and refined over five semesters of a graduate-level software engineering course. The results are then used to provide design recommendations that can generalize to other collaborative project-based learning activities.

Course context and task design

The online cloud computing course is a completely online project-based software engineering course that teaches cloud computing concepts. It is offered to graduate and undergraduate students of Carnegie Mellon University and its worldwide branch campuses. Each semester, between 100 and 200 students participate in the course from across all the campuses. The course is structured around five project-based units. Each unit has several sub-units and culminates in a large individual project that has assessment components corresponding to each sub-unit. The OPE is a real-time, collaborative programming exercise positioned within a sub-unit and is meant to provide individual students with the opportunity to receive feedback and clarify their misconceptions in a collaborative environment before working on their individual projects. Students work in groups of four for the duration of a typical class session (80 min). One of the OPE exercises, as an example, is situated within the first sub-unit of the fourth project unit of the course (Unit 4.1) as shown in Figure 1.

Students complete a pre- and post-test individually before and after the exercise, respectively, to measure individual learning from the collaborative exercise.

For the task design, instructional design best practices recommend that for a given learning goal, instruction must be positioned to explicitly target that learning goal, and the assessment should measure what was targeted (Carver, 2001). Following this recommendation, the overall programming exercise is divided into five different tasks that target five learning objectives (LOs). For each of those learning objectives, two questions each on the pre- and post-tests are assigned to measure student learning from each task. Table 1 shows the learning objectives and examples of pre- and post-test questions corresponding to each subtask, while Figure 1 shows the position of pre- and post-tests within the course.

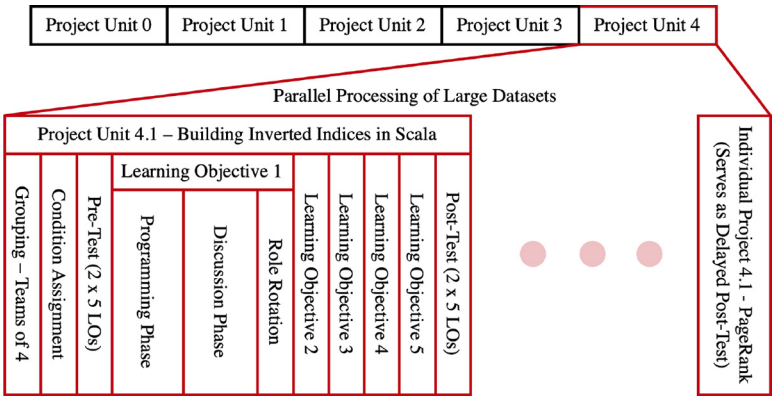

Figure 1.
Course structure, pre-test and post-test alignment

Learning objective	Example pre/post test question (multiple-choice)
1. Writing OS-aware code	What do you need to keep in mind when writing OS-aware code?
2. Need for pre-processing	Whitespace characters are removed while building the index. Why is this necessary?
3. "Map" for word-level count	Which of these is true about combining word counts from different parts of a document?
4. "Reduce" for collating counts within the document	Which Spark code snippet produces an RDD of tuple (word, word frequency)?

Table 1.
Learning objectives, corresponding pre/post test questions (examples)

Completing a task provides the opportunity for students to encounter and practice the content related to an LO and then discuss and reflect on what was learned before work on the next task begins. Each task can, therefore, be considered to consist of two phases – the problem-solving phase, where students participate in the observation, and the discussion phase, where they reflect based on what was just observed. This task structuring, where the exercise is divided into problem-solving tasks, the completion of each of which leads to the opportunity to discuss, can be considered a "macro-script" as described in the script theory of guidance (Fischer *et al.*, 2013). A macro-script sequences the task into learning phases but does not guide behavior within a phase. A "micro-script" on the other hand provides explicit scaffolding for collaborative behavior within each learning phase. Within each task, students are assigned to complementary roles associated with distinct responsibilities that focus on different aspects of the programming task. The roles rotate at the end of each task. This role-assignment and rotation paradigm can be considered a micro-script.

The role assignment paradigm for the OPE is inspired by the industry practice of mob programming (Wilson, 2015; Zuill and Meadows, 2016) and consists of four interdependent roles with well-defined responsibilities that students are assigned to. The *Driver* is the only participant who writes the code, based on high-level instructions received from the *Navigator,* who makes decisions on the next course of action based on discussion with the rest of the group members and communicates that to the Driver to implement into code. The *Researcher* consults resources such as a provided exercise primer and other Web-accessible external support material, as necessary, to assist the team in ideation as well as implementation. The *Project Manager* is responsible for making sure the rest of the team

members are complying with and adequately performing their roles. After each problem-solving phase, the Project Manager leads the group through a reflective discussion in the discussion phase. The roles rotate after each task allowing learners to contribute as well as observe different perspectives and approaches to solving problems in the same session. Finally, this practice requires the externalization of thinking in the course of discussing implementation alternatives, which provides opportunities for knowledge gaps to be revealed and addressed.

Students work on the collaborative programming exercises in the AWS Cloud9 Collaborative IDE. This Integrated Development Environment (IDE) provides a shared editor, terminal, file navigation system and text-chat for collaboratively programming in real-time. The text-chat is embedded with an intelligent conversational agent, built based on the open-source Bazaar framework (Adamson *et al.*, 2014), to provide task instructions, facilitate role-assignment and role-switching and provide conversational prompts, as necessary.

Foundations of designing for learning from collaborative projects

Designing real-time, short collaborative tasks that discourage divide-and-conquer

To address the first challenge of divide-and-conquer, the projects were designed as short, "synchronous" or real-time exercises. This can encourage students to work together to benefit from collaboration. Keeping the exercises relatively short allows students to more easily schedule a time to work together. Working together in real-time then provides them with insights into how other group members implement an idea and in turn, receive feedback on their work. Even within this real-time, collaborative setup, students may resort to divide-and-conquer preventing them from learning from the exercise. The roles focused on different aspects of implementing the given task, therefore, is one explicit way of discouraging this. Only one of the roles, the Driver, is concerned with the implementation while the rest are involved in the decision-making process to determine the implementation path that the group intends to take. This role-assignment requires the entire group to be focused on a single task at a time. Further, arriving at a consensus on the implementation requires students to make their ideas explicit providing opportunities for clarifying misconceptions through discussion and building on group member ideas. The distinction in responsibilities between the roles discourages divide-and-conquer and places an explicit emphasis on the process leading up to the actual implementation including generating different ideas, deciding on one by analyzing the pros and cons, looking up support material for implementation and actually implementing the idea. The students in the roles tasked with deciding or carrying out the actual implementation are "on the spot" but are supported by the rest of the group members in the implementation endeavor.

Role-assignment and rotation paradigms have been used, in general, across a variety of contexts and with different student populations. Early examples include assigning students to the role of summary writer and reviewer for collaborating on a summarization task (O'Donnell and Dansereau, 1992) and the reciprocal teaching paradigm (Brown *et al.*, 1984) where participants are assigned to tutor and student roles to take turns teaching each other concepts from the task. Recent examples such as Process Oriented Guided Inquiry Learning (POGIL) in the context of CS Education (Kussmaul, 2012) also present similar paradigms. The paradigm, owing to the rotation, allows the injection of different perspectives from each of the participants into the task. Further, participants can observe how others would perform the roles they eventually get assigned to and learn from that.

Creating collaboration worthy tasks

In addition to effectively structuring the collaboration to discourage divide-and-conquer, the challenge of designing the exercise to enable students to learn during programming remains. Designing for learning from collaborative exercises can be thought to consist of two components (Schwartz *et al.*, 2016, p. 311). The first is to ensure that the exercise is "collaboration worthy". This means that the exercise must not only be more complex than an individual exercise but also provide many different solution paths that students can decide between through discussion. In the process of choosing a path, students have to make their reasons explicit allowing any misconceptions to be revealed and addressed through the discussion. The second component is the scaffolding of support to reduce the negative effects that might arise from the "conflict" between students suggesting different solution paths and magnify the positive effects of consensus-building and integration of other student's perspectives (conflict-oriented consensus building) (Weinberger and Fischer, 2006).

Prior work on learning from programming shows that programming projects can be a way for students who are unfamiliar with basic programming concepts such as loops, conditionals and function calls, to see those concepts in action (Rivers *et al.*, 2016). However, these tasks may not be "collaboration worthy," as there is often one correct way to implement them in practice. Therefore, we make the OPE tasks collaboration-worthy by focusing on cloud computing concepts taught in the course. Each concept has at least a few different implementations with each providing different advantages in different scenarios.

The OPE exercise has several tasks, each of which is divided into a problem-solving phase and a discussion phase. During the problem-solving phase, group members not in the implementation roles, i.e. the Project Manager and the Researcher, suggest different solutions paths that are potentially conflicting. The Navigator then helps choose one idea from among them by analyzing the pros and cons which the Driver then implements. The roles can, therefore, provide support for conflict-oriented consensus building. The emergence of conflict during the task and support for the resultant consensus-building can be further encouraged through the use of conversational prompts. When presented at opportune times during the problem-solving phase of the task, the prompt reminds students of alternatives that they might not have considered. Prompts presented during the discussion phase also explicitly encourage students to connect the implementation to the learning objective of the task.

Study iterations and results

The design of the OPE described above, was arrived at through several design iterations in semester-long studies conducted in the course.

(1) Study iteration 1 (Sankaranarayanan *et al.*, 2019a) – A preliminary OPE setup was tested against student self-organization where students would work on the same task without the structure provided by the OPE setup.
   - Features: Time-based role rotation with roles changing every 6 min.
   - Group Size: Between three and six students.
   - Roles: Driver (1 student), Navigator (1 student), Mob (1-4 students). The "Mob" role was tasked with brainstorming for ideas about the different solution paths.
   - Results: Student self-organization involved delegating all of the work to one student or dividing and conquering the task, while the OPE setup resulted in distinct collaborative behaviors associated with each role and a more even distribution of labor across them.

(2) Study Iteration 2 (Hilton and Sankaranarayanan, 2019; Sankaranarayanan, 2019) – The same setup was explicitly tested for groups of different sizes (three to six students).

- Results: While the Driver and Navigator roles held up, confusion about the Mob role grew for groups of more than four members. Mob members became less active as they were not clear about what they could contribute that would be distinct from the other Mob members. When it came time to switch roles, it took them more time because they had become less involved in the activity. Larger groups thus spent more time on the activity to achieve the same result.
- Changes for subsequent study: A group size of four was finalized with the Mob role being split into the more explicitly defined roles of "Researcher" and "Discussant". The Researcher is tasked with looking up support material for the group as necessary, and the Discussant is tasked with leading the group discussion to reflect on the learning in the exercise.

(3) Study Iteration 3 (Sankaranarayanan *et al.*, 2019b) compared individual programming to groups with the OPE setup. We hypothesized that individuals would underperform groups with the OPE setup. Even though the group context introduces additional coordination difficulties, the OPE structure would be an effective way to address those while producing positive benefits from the collaboration for the group of students.

- Result: While students assumed their roles in the OPE setup, they were unsure of when to intervene with actions appropriate to their role. Moreover, the Discussant role was also confusing to the students. This prevented the roles from having beneficial effects.
- Changes for subsequent study: "Discussant" role replaced with "Project Manager" who coordinates between group members by roping them in when necessary and ensures role compliance.

(4) Study iteration 4 (Sankaranarayanan *et al.*, 2020a; Sankaranarayanan *et al.*, 2020b) – In the previous studies, large exercises made it hard for students to understand what they were supposed to be learning. We, therefore, structured the exercise into smaller tasks each of which is associated with a learning objective. This is akin to sub-goal labeling which has been shown to reduce the cognitive load during problem-solving in both mathematics and science (Catrambone, 1998; Chi *et al.*, 1989; Margulieux *et al.*, 2018) and increase performance in programming (Margulieux *et al.*, 2012; Morrison *et al.*, 2015). On the pre- and post-tests provided to the students before and after the task, we can then ask questions specific to each learning objective to measure learning not just on the exercise as a whole but also on individual tasks.

- Features:
  - Exercise divided into tasks, each associated with a learning objective.
  - Each task divided into problem-solving and discussion phases to provide dedicated times to work on the task and discuss how it relates to the learning objective before moving on to the subsequent task.
  - The time-based role-rotation was distracting with students being asked to switch roles when they were in the middle of work. The division of the exercise into tasks provided a natural place for the role rotation to happen with students switching roles for each new task.
  - Conversational agent-prompts provided during the discussion phase that explicitly prompted students to connect their work during the task to the learning objective for that task.

- Group size: four students.
- Roles: Driver (one student), Navigator (one student), Researcher (one student), Project Manager (one student).
- Results: Significant learning from the OPE on two of the five tasks with the prompts providing significant benefits over the vanilla OPE in both of those cases. Investigating the *t* tasks where learning did not occur, we found that much closhreeer alignment between the learning objectives and the task was required.

In the latest study iteration, with improved alignment between the task and the learning objectives, we report significant learning improvements for all of the learning objectives.

Recommendations and takeaways

Based on the results from the design studies, we provide some takeaways that may be broadly applicable to online collaborative project-based learning contexts.

- Short real-time exercises – Keeping the duration short makes it easier for students to schedule time. Participating in real-time exercises allows them to observe others doing the work as well as receive feedback on their work in real-time.
- Discouraging divide and conquer through role assignment – By designing roles in an interdependent fashion, the divide and conquer tendencies of students can be discouraged. The roles also require the externalization of various aspects of deciding on an implementation allowing students to surface and clarify misconceptions in the process. Some roles can be designed to be under social pressure to perform while being supported by the remaining roles.
- Collaboration worthy tasks – Tasks have to not only be more complex than those meant for individuals but also should provide many different solution paths that students can decide between through discussion.
- Support for consensus-building – While some roles introduce conflicting ideas for discussion, others can be designed to support the process of consensus-building and integration of other student's perspectives.
- Group size – For the collaborative programming task, a group size of four was decided on based on experimental results. The important idea, however, is to provide enough room for the responsibilities associated with each of the roles to manifest. Too many students might mean some of them will slack and lead to increased coordination difficulties. Too few and not enough conflicting ideas may be discussed to positively impact learning. The exact number of students that can successfully participate in the collaboration will depend on the task and may require some iteration to determine.
- Dividing the exercise into tasks that address specific learning objectives (Sub-goal Labeling) – Breaking the exercise into tasks, each associated with a learning objective allows students to connect what they are working on in the task to what they are supposed to be learning from it. By assigning questions corresponding to each task on the pre- and post-test, we can further measure learning separately by task to understand if students are learning from it. This can help inform the design of the task in the future.
- Discussion in light of an observation – Providing students an opportunity to discuss in light of the problem they just solved allows them to reflect on the task and its connection to the learning objective. We also found that discussion is best situated after

the problem-solving phase of one task and before the next task begins when students are not focused on the problem and can participate completely in the discussion instead.

- Conversational agent prompts to amplify desirable effects – Agent prompts can be used during the task to introduce alternate solution paths that students may not have considered. They can also be used during the discussion phase to help direct student reasoning to the learning objective associated with the task.

References

Adamson, D., Dyke, G., Jang, H. and Rosé, C.P. (2014), "Towards an agile approach to adapting dynamic collaboration support to student needs", *International Journal of Artificial Intelligence in Education*, Vol. 24 No. 1, pp. 92-124.

Brown, A.L., Palincsar, A.S. and Armbruster, B.B. (1984), "Instructing comprehension-fostering activities in interactive learning situations", *Learning and Comprehension of Text*, pp. 255-286.

Carver, S.M. (2001), "Cognition and instruction: enriching the laboratory school experience of children, teachers, parents, and undergraduates", *Cognition and Instruction: Twenty-Five Years of Progress*, Lawrence Erlbaum Associates, pp. 385-426.

Catrambone, R. (1998), "The subgoal learning model: creating better examples so that students can solve novel problems", *Journal of Experimental Psychology: General*, Vol. 127 No. 4, pp. 355.

Chi, M.T., Bassok, M., Lewis, M.W., Reimann, P. and Glaser, R. (1989), "Self-explanations: how students study and use examples in learning to solve problems", *Cognitive Science*, Vol. 13 No. 2, pp. 145-182.

Fischer, F., Kollar, I., Stegmann, K. and Wecker, C. (2013), "Toward a script theory of guidance in computer-supported collaborative learning", *Educational Psychologist*, Vol. 48 No. 1, pp. 56-66.

Hilton, M. and Sankaranarayanan, S. (2019), "Online mob programming: effective collaborative project-based learning", *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 1283-1283.

Kussmaul, C. (2012), "Process oriented guided inquiry learning (POGIL) for computer science", *Proceedings of the 43rd ACM technical symposium on Computer Science Education*, pp. 373-378.

Margulieux, L.E., Catrambone, R. and Schaeffer, L.M. (2018), "Varying effects of subgoal labeled expository text in programming, chemistry, and statistics", *Instructional Science*, Vol. 46 No. 5, pp. 707-722.

Margulieux, L.E., Guzdial, M. and Catrambone, R. (2012), "Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications", *Proceedings of the ninth annual international conference on International computing education research*, pp. 71-78.

Morrison, B.B., Margulieux, L.E. and Guzdial, M. (2015), "Subgoals, context, and worked examples in learning computing problem solving", *Proceedings of the eleventh annual international conference on international computing education research*, pp. 21-29.

O'Donnell, A.M. and Dansereau, D.F. (1992), "Scripted cooperation in student dyads: a method for analyzing and enhancing academic learning and performance", *Interaction in Cooperative Groups: The Theoretical Anatomy of Group Learning*, pp. 120-141.

Rivers, K., Harpstead, E. and Koedinger, K.R. (2016), "Learning curve analysis for programming: which concepts do students struggle with?", *ICER*, pp. 143-151.

Sankaranarayanan, S., Wang, X., Dashti, C., An, M., Ngoh, C., Hilton, M., Sakr, M. and Rosé, C. (2019b), "An Intelligent-agent facilitated scaffold for fostering reflection in a team-based project course", *International Conference on Artificial Intelligence in Education*, Springer, Cham, pp. 252-256.

Sankaranarayanan, S. (2019), "Online mob programming: effective collaborative project-based learning", *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, pp. 1296-1296.

Sankaranarayanan, S., Kandimalla, S., Hasan, S., An, H., Bogart, C., Murray, R., Hilton, M., Sakr, M. and Rosé, C. (2020a), "Creating opportunities for transactive exchange for learning in Performance-Oriented team projects", *International Conference of the Learning Sciences*.

Sankaranarayanan, S., Kandimalla, S., Hasan, S., An, H., Bogart, C., Murray, R., Hilton, M., Sakr, M. and Rosé, C. (2020b), "Agent-in-the-loop: conversational agent support in service of reflection for learning during collaborative programming", *International Conference on Artificial Intelligence in Education*.

Sankaranarayanan, S., Wang, X., Dashti, C., An, H., Ngoh, C., Hilton, M., Sakr, M. and Rosé, C. (2019a), "Online mob programming: bridging the 21st century workplace and the classroom", *International Conference on Computer-Supported Collaborative Learning*, pp. 855-856.

Schwartz, D.L., Tsang, J.M. and Blair, K.P. (2016), *The ABCs of How we Learn: 26 Scientifically Proven Approaches, How They Work, and When to Use Them*, WW Norton and Company.

Weinberger, A. and Fischer, F. (2006), "A framework to analyze argumentative knowledge construction in computer-supported collaborative learning", *Computers and Education*, Vol. 46 No. 1, pp. 71-95.

Wilson, A. (2015), "Mob programming-what works, what doesn't", *International Conference on Agile Software Development*, Springer, Cham, pp. 319-325.

Zuill, W. and Meadows, K. (2016), "Mob programming: a whole team approach", *Agile 2014 Conference, Orlando, FL*, Vol. 3.

Author affiliations

Sreecharan Sankaranarayanan, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Siddharth Reddy Kandimalla, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Mengxin Cao, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Ignacio Maronna, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Haokang An, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Chris Bogart, Institute for Software Research, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

R. Charles Murray, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Michael Hilton, Institute for Software Research, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Majd Sakr, Department of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Carolyn Penstein Rosé, Language Technologies Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

Corresponding author

Sreecharan Sankaranarayanan can be contacted at: sreechas@andrew.cmu.edu