This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

Collaborative Programming for Work-Relevant Learning: Comparing Programming Practice with Example-Based Reflection for Student Learning and Transfer Task Performance

Sreecharan Sankaranarayanan, *Student Member, IEEE,* Siddharth Reddy Kandimalla, Christopher Bogart, R. Charles Murray, Michael Hilton, Majd Sakr, Carolyn Rosé, *Senior Member, IEEE*

***Abstract*— Computer science pedagogy, especially in the higher education and vocational training context, has long-favored the hands-on practice provided by programming tasks due to the belief that this leads to better performance on hands-on tasks at work. This assumption, however, has not been experimentally tested against other modes of engagement such as worked example-based reflection. While theory suggests that example-based reflection could be better for conceptual learning, the concern is that the lack of practice will leave students unable to implement the learned concepts in practice, thus leaving them unprepared for work. In this paper, therefore, we experimentally contrast programming practice with example-based reflection to observe their differential impact on conceptual learning and performance on a hands-on task in the context of a collaborative programming project. The industry paradigm of Mob Programming, adapted for use in an online and instructional context, is used to structure the collaboration. Keeping with the prevailing view held in pedagogy, we hypothesize that example-based reflection will lead to better conceptual learning but will be detrimental to hands-on task performance. Results support that reflection leads to conceptual learning. Additionally, however, reflection does not pose an impediment to hands-on task performance. We discuss possible explanations for this effect, thus providing an improved understanding of prior theory in this new computer science education context. We also discuss implications for the pedagogy of software engineering education, in light of this new evidence, that impacts student learning as well as work performance in the future.**

***Index Terms*— Worked examples, Computer science education, Project-based learning, Collaborative learning tools, Computer-supported collaborative learning, Conversational agent-based support, Mob programming, Collaborative programming, Ensemble programming.**

I. Introduction

PROGRAMMING practice holds a preeminent place in the teaching and learning of computer science. As a field that values practitioners' ability to implement solutions to hands-on tasks, this bias is understandable. Learning science literature, such as that on example-based learning [1], however, explicitly compares problem-solving practice with worked example-based learning, in domains such as mathematics and science, to conclude that worked example-based reflection is more effective for learning, especially in novice learners. Following from this theory, programming practice, being the analogue for problem-solving in the computer science context, could be hypothesized as being inferior to worked example-based reflection from the standpoint of student learning. The concern voiced by educators is that while it may indeed be the case that worked example-based reflection helps students learn the conceptual aspects of programming better, they may be left less able to implement those learned concepts in actual hands-on programming tasks, owing to their lack of hands-on practice. Without experimental analyses to offer definitive evidence one way or the other, therefore, prevailing dogma dictates pedagogy, leaving programming practice as the preferred way to help students learn as well as implement what they have learned in hands-on tasks.

In order to better understand the application of the literature on example-based learning to computer science education while influencing pedagogy to move in a direction better suited to prepare students for work, we present, in this paper, an experimental study with a study design that allows us to

Manuscript received July 01, 2021; revised March 01, 2022; accepted xxx. Date of publication xxx; date of current version March 01, 2022. This work was supported in part by the U.S. National Science Foundation under grants IIS 1822831, IIS 1917955, and funding from Microsoft. *(Corresponding author: Sreecharan Sankaranarayanan.)*

S. Sankaranarayanan, was with the Language Technologies Institute, Carnegie Mellon University, Pittsburgh PA 15213 USA. He is now with Amazon.com, Inc., Seattle WA 98109 USA (e-mail: sreechas@cs.cmu.edu).

S. R. Kandimalla, is with the Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: skandima@cs.cmu.edu).

C. Bogart, is with the Institute for Software Research, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: cbogart@cs.cmu.edu).

R. C. Murray, is with the Language Technologies Institute, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: rcmurray@cs.cmu.edu).

M. Hilton, is with the Institute for Software Research, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: mhilton@cs.cmu.edu).

M. Sakr, is with the Department of Computer Science, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: msakr@cs.cmu.edu).

C. P. Rosé, is with the Language Technologies and Human Computer Interaction Institutes, Carnegie Mellon University, Pittsburgh PA 15213 USA (email: cprose@cs.cmu.edu).

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

compare different levels of programming practice and worked example-based reflection to understand their differential impact on conceptual learning from the task as well as performance on a hands-on transfer task.

We situate the study in a work-relevant graduate-level course on Cloud Computing offered online at Carnegie Mellon University's campuses in Pittsburgh, Silicon Valley, and Qatar. The course is considered a work placement course in that it is aimed at helping successful students transition to careers in Cloud Computing such as Software Engineering for the Cloud and System Administration. This study lays the foundation for the adoption of this application across a full spectrum of technical courses in other work-relevant contexts including community college and corporate learning.

At the outset, we do not question the value of problem-solving practice for learning by doing software development. Instead, we seek to compare its strengths and weaknesses with that of an alternative instructional activity type shown to be effective in other contexts – worked example-based reflection. The literature on example-based learning [1] compares extensive problem-solving practice with learning from worked out examples that consist of the givens of a problem, solution steps, and the final solution itself [2]. This literature establishes that problem-solving practice is generally inferior to worked example study, especially when the learners are considered novices with respect to the concepts they are learning. Even so, the integration of worked example study into the computer science curriculum has been stifled by the belief that it will leave students less able to engage in subsequent authentic problem-solving tasks from lack of hands-on practice. While some computer science education research challenging this assumption has started to emerge [3], the published studies are mainly focused on the introductory curricular context. This leaves open the question of identifying the right trade-off between problem-solving practice and worked example study for learning and performance in the advanced, more work-relevant computer science curriculum.

When we move from the introductory to the advanced curricular context, additional considerations for the design of such learning activities also start to emerge. Specifically, as workplace software engineering practices evolve to include more team-based and collaborative activities [4], the need for collaborative learning opportunities, especially in courses that are more vocational than foundational, becomes ever more apparent. Consequently, we choose to incorporate the study in a collaborative programming project. The collaborative programming project is scaffolded with the use of conversational agent-based prompts based on an industry paradigm for collaborative programming called Mob Programming that was adapted for use in an online instructional context in prior work [5] – [8].

Starting with the study presented in this paper, we establish a research agenda investigating the trade-off between problem-solving practice and worked example study in collaborative programming activities offered at-scale in advanced courses in the computer science curriculum. The goal is to find the right combination of the two to maximally impact learning from the task with minimum detriment to performance on subsequent authentic problem-solving. With this, we aim to contribute to the literature on example-based learning and inform the design of collaborative programming exercises for learning and hands-on task performance.

In the remainder of this paper, we start by developing an understanding of prior theoretical work in example-based learning as well as computer science education to come to a nuanced understanding of the differences between worked example-based reflection and problem-solving practice, and where they work well. In the Methods section that follows, we first describe the context in which the study was conducted and the characteristics of the participants. We then describe the design of the collaborative learning activity and scaffolding in detail. This is followed by the description of the study design that enables us to compare the differential effects of varying levels of problem-solving practice and example-based reflection on student's conceptual learning from the task and performance on a hands-on transfer task. This is followed by our Hypotheses, predominantly motivated by prevailing pedagogical practice which tilts heavily toward problem-solving practice in the form of programming. Evidence that supports or rejects our hypotheses is presented in the following Analysis and Results section. In the Discussion section that follows, we expand on the results and their implications for theory of example-based learning and computer science pedagogy. We then conclude by summarizing the findings charting a course for future work.

II. Theoretical Foundation and Motivation

Problem-solving tasks in the form of computer programming exercises have held an exalted position in the computer science curriculum both for courses taught in-person and at-scale. Two possible reasons can be identified for their preeminence in the curriculum. First, these tasks are seen as "authentic" in their similarity to the day-to-day work of software engineers. Second, their place in the curriculum is informed by computer science education research that focuses overwhelmingly on the introductory computer science context [9]. The value of problem-solving practice for learning software development by doing has been recognized by this research. However, a more nuanced view may be necessary to separate places in the curriculum where problem-solving practice can have the most impact from places where alternative methods of engagement, such as worked example study, might be better.

Early research in computer science pedagogy [10] – [12] points to the separation of programming knowledge into the syntax and semantics of programming language constructs, and its conceptual aspects such as the mechanisms and explanations that are used to compose solutions. In other words, students learning programming must learn the conceptual aspects such as variables, loops, iteration, and functions, in addition to the procedural aspects of converting these concepts into working code. It is possible to hypothesize, therefore, that at places in the curriculum where students have gained enough procedural knowledge to implement concepts once they are learned, their time is better spent learning

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

concepts that they are new to. Cognitive Load Theory [13] can help us understand the value that can be gained from driving the limited cognitive resources that students have towards conceptual learning from a task.

A. Cognitive Load Theory and Example-Based Learning

Cognitive Load Theory points out that the identification and induction of schemata, i.e., domain-specific knowledge structures [14], from problem states is the primary function of conceptual learning from a task. Since problem-solving practice may involve superfluous production steps that place a load on the limited cognitive resources of a student, these resources are taken away from the processes that could most impact their conceptual learning. The literature on example-based learning [1] builds on this to investigate the differential impact of problem-solving practice and worked example study on conceptual learning from a task. This literature establishes that problem-solving practice is generally inferior to worked example study precisely because of the reason outlined above. The worked examples do away with the superfluous production steps to allow students to completely focus on learning from the problem states showcased. We can see an isomorphic relationship between the processes of schema acquisition and automation pointed to in the worked example literature [15] and the separation of programming knowledge into its conceptual and procedural aspects pointed to in the computer science education literature [10] – [12]. This would indicate that problem-solving practice in the form of programming exercises may not be the most efficient for conceptual learning and worked example study can play that role better. Indeed, computer science education research has also started to gain this nuanced understanding of the separation between the conceptual and procedural aspects of programming knowledge. The preeminence of problem-solving practice in the curriculum has started to be challenged as a result. Alternative methods of engagement such as the use of two-dimensional Parson's problems [16] which are a type of code completion problem useful for teaching syntactic and semantic language constructs i.e., the procedural aspects of programming, the CMX game-based learning environment, also targeting the procedural aspects of programming knowledge, such as arrays [17] and worked example study [18] targeting the conceptual aspects have started to emerge. Even so, these alternative methods of engagement are largely absent as students move from the introductory context and on to courses advanced in the computer science curriculum.

In the advanced computer science curriculum, the preeminence of problem-solving practice becomes harder to justify as students have gained substantial foundational knowledge in the procedural aspects of programming but are new to advanced concepts such as Artificial Intelligence, Cloud Computing, and Data Science that are taught in these courses. Students can gain more value, therefore, from focusing more of their time on conceptual learning. Regardless, the suggestion that worked example study ought to be better integrated into the advanced computer science curriculum has remained controversial [19]. Since example-based learning involves less hands-on work to convert conceptual knowledge into actual programs, the concern is that while students may learn more, they may be left less able to implement their knowledge in projects that follow their learning experience. In the introductory computer science context, this concern may be warranted because students are new to both the conceptual, as well as the procedural aspects of programming. It is less warranted, however, in places in the advanced curricular context where students are novices with respect to the concepts taught and can therefore gain more from focusing on conceptual knowledge. Without confirmatory evidence, however, pedagogical practice continues to follow prevailing dogma. Therefore, the research agenda being pursued in this paper becomes warranted for two reasons. First, to provide confirmatory evidence of the application of the literature on example-based learning to a new domain, i.e., advanced computer science education, predicated on the nuanced understanding of the separation of programming knowledge into its conceptual and procedural aspects posited here. Second, to provide evidence that will inform pedagogy that will best prepare students for work. We thus present a research agenda that explicitly investigates the right combination of problem-solving practice and worked example-based reflection for impacting student learning and hands-on task performance at-scale in the context of advanced courses in the computer science curriculum becomes called for.

B. Need for Collaborative Learning

Since the advanced curricular context is more vocational than foundational, the need for learning tasks to be "authentic" in their similarity to the day-to-day work of software engineers is indeed important. Workplace software engineering practices are also evolving to include more team-based and collaborative activities [4]. Consequently, the design of these tasks as individual experiences for students starts to come into question as well. While the conceptual underpinnings of the comparison between problem-solving practice and worked example study remain the same, barring some notable exceptions [20] this comparison has not been explicitly investigated in a collaborative learning context. In this paper, therefore, we further the research agenda outlined above by presenting a study situated in a synchronous collaborative learning activity offered as a part of an advanced online course in the computer science curriculum. By varying the boundary between problem-solving practice and worked example discussion during the activity, we are able to compare students' learning from the task, and hands-on performance to determine the combination most suited for this context. Analyzing the results with respect to the characteristics of the context in which the study was conducted, we can further the research agenda by discussing implications and directions for future research.

III. METHOD

A. Course Context and Participants

This study was conducted in the Fall 2020 semester-long offering of a graduate-level project-based online course covering Cloud Computing concepts offered to graduate and senior undergraduate students belonging to a large North

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

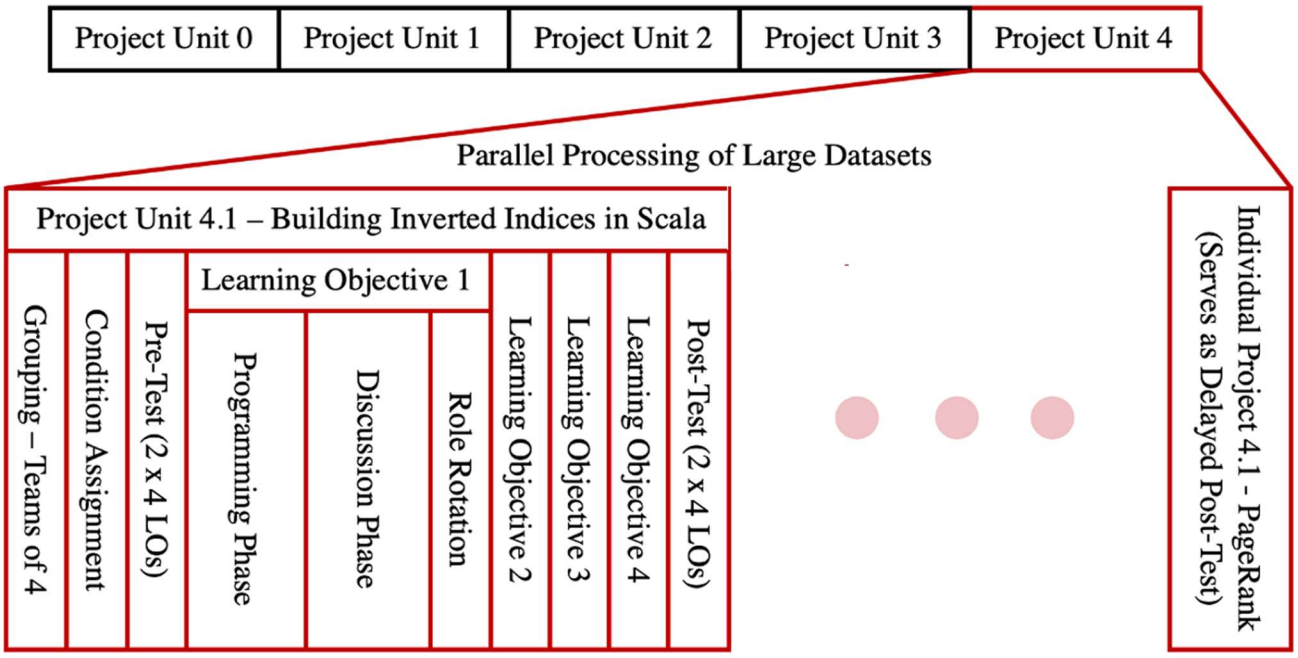

Fig. 1. Course structure, pre-test, post-test, and delayed post-test alignment

American university and its branch campuses[1]. The course is structured around four project-based units. Each unit has several sub-units and culminates in a large individual project that has assessment components to evaluate achievement in each sub-unit. Our experiment is situated within the first sub-unit of the fourth project unit of the course that focuses on "Building Inverted Indices in Scala". In preparation for the individual project in this unit, students, in groups of 4, work with our synchronous (real-time) collaborative programming activity, called the Online Programming Exercise (OPE). In this exercise, students have access to a collaborative programming environment called Cloud9[2] that is instrumented with a text-chat that they can use for communication during programming. The exercise is immediately preceded and succeeded by a pre- and post-test respectively. Performance improvement on the post-test over the pre-test serves as a measure of conceptual learning, while performance on the complex individual software development project component corresponding to this sub-unit serves as a procedural and conceptual problem-solving transfer task (delayed post-test). A summary of the course structure and the location of the study within it is shown in Fig. 1. A total of 48 students participated in the exercise and the subsequent project from across two campuses.

The unit introduces students not only to the conceptual aspects of inverted indices and the PageRank algorithm, but also to the procedural aspects of thinking in, and implementing concepts learned in a new programming language, Scala. Scala uses a programming paradigm called functional programming that most students are unfamiliar with prior to this unit. Consequently, this becomes the ideal context to test the differential impact of problem-solving practice, in the form of collaborative programming, and worked example-study, in the form of collaborative reflection, on learning from the task, and performance on the subsequent problem-solving transfer task.

Since the course has been offered online for over 15 semesters, the course content and structure did not need to be changed in response to the move to online learning in the wake of the COVID-19 pandemic. The main effect was seen in the enrollment numbers which were about half the usual. Some students participated from off-campus locations in other time zones. Options of time slots were provided to students for participating in the synchronous collaborative exercise such that each time slot was amenable to students from different time zones.

B. Collaborative Task Design

The design of the two elements of the collaborative task i.e., collaborative reflection phase and collaborative programming phase are described below.

In the collaborative reflection phase, students are presented with a worked-out example. Since the problem has already been worked out, their time is engaged instead in collaborative reflection. Learning from worked examples hinges on being able to draw students' attention to the relevant problem states while helping them navigate away from superfluous ones [19]. In prior work, this has been achieved by using various means such as classification of examples by common schema [21] contrasting examples, and prompt-directed self-explanation [22]. In our study, we use conversational agent-based prompt scaffolding during to direct students' joint attention to the aspects of the problem state that are relevant to their learning. The collaborative learning context might not only allow students to rely on the experience of all the members of the team to identify relevant problem states but, when designed appropriately, can require students to make their reasoning

[1] http://www.cs.cmu.edu/~msakr/15619-f20/

[2] https://aws.amazon.com/cloud9/

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

TABLE I
LEARNING OBJECTIVES, AND EXAMPLES OF CORRESPONDING PRE/POST TEST QUESTIONS AND COLLABORATIVE REFLECTION PROMPTS

#	Learning Objective	Example Pre/Post-Test Question (Multiple-Choice)	Example Collaborative Reflection Prompt
1	Utilize the suitable map and reduce operations in Spark to transform and aggregate data.	Examine the following ways of computing the sum by key of the exampleRDD. Which of them is the most efficient?	Think of the solution you were attempting. Is this the same as what the bot presented in the chat? Take this time to explain the solution to each other.
2	Utilize suitable functional operations in Scala to transform data structures.	Which of the following is a right way of transforming srcList to destList?	There are a few different ways of doing element indexing such as dot indexing or the scala case() function. Which of these is shown in the example and which would be better aligned with best practice?
3	Identify the use case of a JOIN operation and utilize JOIN to combine and transform different RDDs in Spark.	Calling JOIN on two RDDs of type (K, V1) and (K, V2) results in which of the following types?	How is the combining of values implemented here? Can you unpack the working based on the example presented?
4	Identify the use case of the cache() method and utilize it to improve the performance of a Spark application.	What is caching in memory meant to achieve in Spark?	What factors do you need to keep in mind while deciding what to cache? See if you can discover the list of factors from the example.

TABLE II
THE BOUNDARY BETWEEN COLLABORATIVE PROGRAMMING PRACTICE AND WORKED EXAMPLE DISCUSSION ACROSS TASKS.

Task #	Mostly Problem-Solving	Equal	Mostly Worked Example Reflection	Only Worked Example Reflection
1	12 + 6	9 + 9	6 + 12	0 + 18
2	12 + 6	9 + 9	6 + 12	0 + 18
3	8 + 4	6 + 6	4 + 8	0 + 12
4	4 + 2	3 + 3	2 + 4	0 + 6

Legend: a + b, where 'a' indicates time spent on collaborative programming, and 'b' represents the time spent on collaborative reflection.

explicit. One student articulating their idea then provides an opportunity for other students in the team to challenge, extend, or integrate that idea with their own, resulting in transactive exchange that is associated with learning [23]. Indeed, prior work in other contexts has provided evidence for the strategic use of conversational agent-based prompts to productively impact student behavior for supporting their learning [24].

For structuring the collaboration in the problem-solving portion of the activity, we turn to a collaborative programming paradigm used in the industry called Mob Programming [25] – [26]. Structuring collaboration is necessary as research has shown that students do not utilize collaborative learning opportunities without intentional design [27]. Further, a structured collaboration prevents the possible breakdown of group processes. A well-structured collaborative problem-solving portion of the intervention can serve as a strong baseline for the comparison. Based on the Mob Programming paradigm used in the industry, prior studies have designed roles for use in online instructional contexts [5] – [8] Students are assigned to roles that are tasked with different aspects of problem-solving with the roles rotating over the course of the task. Well-defined responsibilities for each student prevents group process breakdown, while the interdependent nature of the roles ensures collaboration for problem-solving. The roles are described further in the following section.

1) Role Scaffolding

During the problem-solving phase, students are assigned to four interdependent roles designed based on the industry

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

practice of Mob Programming. Mob Programming involves a team of co-located developers assuming interdependent roles to work synchronously on a single problem, on one computer. For our task, students are assigned to the following four roles that have been adapted for an online instructional context in prior work [5] – [8].

a) ***The Driver*** *writes the code based on a high-level consensus decision of the team arrived at by the Navigator.*

b) ***The Navigator*** *arrives at the team consensus on the next course of action based on a discussion with the rest of the team members and communicates that to the Driver to implement into code. The Driver and Navigator are assisted in this endeavor by students in two supporting roles.*

c) ***The Researcher*** *consults resources such as a provided task primer and other web-accessible external support material, as necessary, to assist the team in idea-generation as well as implementation.*

d) ***The Project Manager*** *is responsible for making sure the rest of the team members are complying with and adequately performing their roles.*

The roles rotate after each task such that each team member gets to play the Driver role once during the exercise. Knowing that the roles will rotate encourages all participants, including those in the support-oriented roles, to pay attention in case they are called on to work on the implementation for the next task. The role rotation also provides an opportunity for individuals to contribute in various capacities over the course of the exercise. The separation of responsibilities between roles requires the externalization of thinking while discussing implementation alternatives, which provides opportunities for knowledge gaps to be revealed and addressed.

C. Instructional Design

For instrumenting the collaborative task to best support student learning, we turn to instructional design best practices. These practices are about aligning the learning goal with instruction positioned to explicitly target that goal, and the assessment that measures what was targeted [28] –[29]. Actualizing this best practice, the collaborative programming exercise is divided into four tasks, each targeting a learning objective (LO). Each task is divided into a problem-solving phase where students, assigned to four interdependent roles, work on the programming problem, and a collaborative reflection phase where they are guided by conversational agent-based prompts to reflect based on a presented worked-example. Completing a task provides the opportunity for students to encounter and practice content related to the learning objective and then collaboratively reflect on what was learned before work on the next task begins. Students are assigned to four interdependent roles during the problem-solving phase by the conversational agent and the collaborative reflection is guided using conversational agent-based discourse-level prompts. Table I shows examples of these collaborative reflection prompts. While the focus of the prompts may seem to be procedures specific to the programming language, they are meant to evoke a conceptual discussion in response. As an example – the entry in the second row "dot indexing versus the scala case() function " is meant to evoke a discussion about different ways indexing can happen and which one turns out to be more efficient. Another way to think about the separation between the conceptual and the procedural is that once the learner has understood this difference, they will be able to identify which method is more efficient even in a completely new programming language, once they learn what each relevant procedure in that language does. The conceptual learning, therefore, is independent of the programming language paradigm i.e., the procedural aspects of implementing the concept, once learned.

D. Measurement

Before and immediately after the task, students take a pre- and post-test respectively, where two multiple-choice questions are assigned for each learning objective. The pre-test score and post-test score for each LO is the average of the scores of the two questions targeting that LO, and thus ranges between 0 and 1. Performance improvement from pre- to post-test is used as a measure of students' conceptual learning from the task. Students subsequently work on an individual programming project. Since this project involves implementing the concepts learned from the collaborative exercise, it can serve as a delayed measure of conceptual as well as procedural knowledge. The project contains six tasks that map directly to the learning objectives of the collaborative activity. Table I shows the learning objectives and examples of pre- and post-test questions corresponding to each task, while Fig. 1. shows the position of pre-, post-, and delayed post-tests within the course.

E. Study Design

In the weeks before the synchronous collaborative activity in which the study was conducted, students first read text and watched video content explaining the activity, the collaborative programming environment, and the roles that they were to be assigned to. After that, students, in groups of 4, participated in a pilot activity that mimicked the activity in which the actual study would be conducted from end-to-end. The content for this pilot activity was kept relatively lightweight so that the focus could remain on helping students familiarize themselves with the collaborative programming environment and interacting in their roles. For both the pilot, and the study, groups were formed based on students' time-availability that they were polled about earlier in the course.

For the activity where the study took place, students, in groups of 4, were assigned to 4 different conditions. The conditions were as follows –

*1) **Mostly problem-solving***

Students spend two-thirds of the time on collaborative problem-solving before being cut-off and presented with a worked example solution for collaboration reflection.

*2) **Equal problem solving and worked example discussion***

Students spend equal amounts of time on collaborative problem-solving and reflection based on the presented worked example.

*3) **Mostly worked example discussion***

Students spend one-third of the time on problem-solving and the rest of collaborative reflection.

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

*4) **Only worked example discussion***

Students spend all of the time reflecting based on the presented worked example.

The total activity duration was 80 minutes. 5 minutes in the beginning were allocated for setup and helping students familiarize themselves with each other. Another 5 minutes were allocated at the end of the activity for students to submit their solution. 10 minutes was the buffer allocated for context-switching from problem-solving to worked example discussion and rotating roles. The remaining 60 minutes was for the actual tasks which was split according to the conditions that students were assigned to as shown in Table II. A few minutes in the beginning and end were used for becoming acquainted with teammates and submitting the completed solution respectively. A total of 48 students participated in the activity in groups of 4 with 3 groups students i.e., 12 students in each of the four conditions.

OPE_Bot: "Ok, we are switching roles now. The new task is unlocked."

OPE_Bot: "The new roles are -
S1 - Navigator
S2 - Driver
S3 - Researcher
S4 - Project Manager"

S3 (Researcher): "@S1 (Navigator), what could (sic) I help you find?"

S1 (Navigator): "Are we supposed to use only the helper functions?"

S3 (Researcher): "Yes, I think so! Looked at it (the task writeup."

...

S4 (Project Manager): "@S2 (Driver), want to try implementing this in that block with our variables?"

S2 (Driver): It has some build errors.

....

Fig. 2. Example student interaction facilitated by the conversational agent during the collaborative programming phase.

An example student interaction facilitated by the conversational agent during the collaborative programming phase is shown in Fig. 2. An example interaction during the collaborative reflection phase is shown in Fig. 3.

OPE_Bot: "You have run out of time for this test case. Don't worry! Here is a reference solution - "

OPE_Bot: .... (Reference Solution)

OPE_Bot: "Can you compare the solution you were attempting to the example shown?"

S1: "Oh it uses join."

S2: "Yes..." (Still viewing the example)

S3: "and it uses pattern matching here"

...

S4: "I guess zip should work too?"

S1: "Yeah I think so."

S1: "here if we use indexing it would a bit confusing I guess" (sic)

...

OPE_Bot: Based on your solution and the example, which implementation is better aligned with best practice?

S1: "Pattern matching is more readable."

S1: "Here we use join to combine the values, we also tried zip and we think zip would work too"

OPE_Bot: When done identifying the explanation, write the explanation agreed on in the chat.

S3: "We chose join to combine values and pattern matching because it is more readable"

OPE_Bot: "Ok, we are switching roles now. Your next task is unlocked."

Fig. 3. Example interaction in the collaborative reflection phase.

IV. HYPOTHESES

*1) **Hypothesis 1**: The Online Programming Exercises (OPEs) result in pre- to post-test learning gains*

Regardless of whether students participate in more problem-solving or worked example discussion, the activity results in pre- to post-test learning gains.

*2) **Hypothesis 2**: Worked example discussion contributes more to conceptual learning as measured by the pre- to post-test learning gains than problem solving.*

Thus, we expect the most learning where students maximize time on worked example study, and less learning as students spend more of their time on problem solving.

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

TABLE III
LEARNING OBJECTIVES, AND EXAMPLES OF CORRESPONDING PRE/POST TEST QUESTIONS AND COLLABORATIVE REFLECTION PROMPTS

#	Learning Objective	Example Pre/Post-Test Question (Multiple-Choice)	Example Collaborative Reflection Prompt
1	Utilize the suitable map and reduce operations in Spark to transform and aggregate data.	Examine the following ways of computing the sum by key of the exampleRDD. Which of them is the most efficient?	Think of the solution you were attempting. Is this the same as what the bot presented in the chat? Take this time to explain the solution to each other.
2	Utilize suitable functional operations in Scala to transform data structures.	Which of the following is a right way of transforming srcList to destList?	There are a few different ways of doing element indexing such as dot indexing or the scala case() function. Which of these is shown in the example and which would be better aligned with best practice?
3	Identify the use case of a JOIN operation and utilize JOIN to combine and transform different RDDs in Spark.	Calling JOIN on two RDDs of type (K, V1) and (K, V2) results in which of the following types?	How is the combining of values implemented here? Can you unpack the working based on the example presented?
4	Identify the use case of the cache() method and utilize it to improve the performance of a Spark application.	What is caching in memory meant to achieve in Spark?	What factors do you need to keep in mind while deciding what to cache? See if you can discover the list of factors from the example.

3) ***Hypothesis 3****: Programming practice contributes most to ability to program.*

Thus, we expect students who practice more at writing code to perform better on a subsequent authentic programming task than students who spent more time on worked example discussion.

V. ANALYSIS AND RESULTS

1) ***Hypothesis 1****: Students learned in all conditions.*

We first tested to ensure that there was significant pre- to post-test learning in evidence across conditions. For this analysis, we built an ANOVA model with Time Point (pre- vs post-), Learning Objective, Group, and Condition as well as all two- and three-way interactions of Time Point, Learning Objective, and Condition as independent variables, and the test score as the dependent variable. There was a main effect of time point such that test scores were higher on average at post-test time $F(1, 371) = 1.51$, $p < .0003$, effect size .21 s.d. None of the two- or three-way interactions were significant. Therefore, students learned for all learning objectives in all conditions between the pre- and post-test. Thus, Hypothesis 1 is fully supported.

2) ***Hypothesis 2****: Worked example study contributes more to conceptual learning as measured by the pre- to post-test learning gains than problem solving.*

In order to test the second hypothesis, we computed an ANCOVA model with Condition and Learning Objective as independent variables, Group nested within Condition, pre-test score as a covariate, and post-test score as the dependent variable. When we treat Condition as a nominal variable with 4 levels, there is no significant effect of condition. However, if we transform the Condition variable to simply compare the no programming condition to the others, there is a significant effect in favor of no programming $F(1,171) = 4.2$, $p < .05$, effect size .18, with an advantage for no programming. For a finer grained analysis, we checked the relationship between percentage of time spent on problem solving and learning and found a significant negative correlation. For this analysis, we built an ANCOVA model with post-test score as the dependent variable, pre-test score and percentage of time spent on problem solving as covariates, and Learning Objective as an independent variable. In this analysis, there was a significant negative correlation between percentage of time spent on problem solving and post-test score in this analysis, indicating that as we manipulated amount of time spent on problem solving, the more time spent on problem solving, the less learning took place, $F(1, 182) = 5.42$, $p < .05$. The partial correlation accounting for just the effect of percentage of time spent on problem solving is $R=.17$. Thus, Hypothesis 2 is supported, though the effect may be weak.

3) ***Hypothesis 3****: Programming practice contributes most to ability to program.*

In order to test the third hypothesis, we constructed a repeated measures dependent variable that is comprised of the 6 separate scores assigned to evaluate the quality of the authentic problem-solving task. The Construct variable distinguished between the 6 different criterion scores. We built an ANCOVA model with pre-test score and percentage of time spent on problem solving as covariates and Construct as the independent variable. The criterion score for each of the Construct labels was the dependent variable. There was no effect of Construct or Condition. We also tested the model with percentage of time nested within Construct in case the condition

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

variable had a differential effect across construct levels, but there was still no effect. Thus, Hypothesis 3 was not supported. What this means is that not only did students learn more conceptual knowledge from spending more time studying worked examples, but they did also not incur any deficit in terms of their ability to perform on the authentic task, either overall, or on any single criterion score.

The verbose results from the analyses are shown in Table III.

VI. Discussion

We find, interestingly, that the condition where students spent all of their time on collaborative reflection based on the presented worked example was best for conceptual learning from the task and did not incur any deficit in terms of their ability to perform on the authentic task. The finding about conceptual learning impact is in line with what the literature on example-based learning would predict. For maximally impacting conceptual learning from the task, it is best to do away with superfluous production steps that might draw student attention away from conceptual learning and focus all of their time instead on highlighting problem states relevant to conceptual learning. In the synchronous collaborative learning task that students participated in as a part of this study, this was achieved using a two-fold mechanism. In addition to simply being presented with a worked example, conversational agent prompts were used to direct student attention to the problem states relevant to their conceptual learning. In response to these prompts, students, in their groups were required to direct their attention to finding and discussing the relevant problem states. The discussion meant that any misconceptions could be revealed and addressed for the group as a whole. An example of such an interaction in response to the collaborative reflection prompt was shown in Fig. 3.

The finding about the lack of differential impact on authentic task performance is interesting. Since students are new to the procedural aspects of writing working programs in the new language, Scala, that is introduced in this unit, we hypothesized that hands-on practice would serve students better for performance on the authentic task. However, there turned out to not be a significant difference. When we asked students to understand why this might be the case, we found that most students were comfortable learning the syntactic and semantic aspects of programming, even in a new language, all on their own. Additionally, some foundational knowledge they had gained from working on other programming languages with a similar paradigm was useful in orienting them towards learning relevant syntactic and semantic aspects in this new language. Students preferred that their time in the collaborative activity was instead spent on learning concepts that they find harder to learn on their own. It should be noted, however, that the intervening week between the end of the collaborative activities and when the project was due could have served to wash out some of the differential impact. Since students said that they are comfortable searching for syntactic and semantic issues on their own, this could indeed have been the case. It is possible, however, that students who received less problem-solving practice spent more time searching, and less time on the implementation while those who participated in worked example study had to do more of the heavy lifting around implementation themselves. This can only be confirmed by a process analysis of the data about how students split their time while working on the individual project. Since this study was not instrumented to collect this data, this is a caveat for the findings, and a factor to investigate in future work.

Even so, the evidence presented in this study starts to make it possible to recommend better integrating worked example study into the advanced computer science curriculum. While the value that hands-on software development brings to learning software engineering is not disputed, a nuanced view of the separation of programming knowledge into its procedural and conceptual components was used for isolating the impact on conceptual learning in this study. We see clearly the role of conceptual learning in advanced computer science and the value of reflection over practice for this learning. At the same time, we see that maximizing hands on work is not necessary for supporting needed learning of the hands-on skills of computer programming. The evidence supports a view that shifting more time towards reflection would be beneficial. More work to pinpoint the ideal balance and placement of reflection versus practice is still needed.

The study also presented a way to actualize the theory of example-based learning in a collaborative learning environment at scale using conversational agent-based prompts to take advantage of the different perspectives that could be contributed by the students in the text-chat. Based on these results and the additional questions that the findings have raised, future work to further this research agenda is warranted.

VII. Conclusion and Future Work

In this paper, we present first fruits of a research agenda investigating the trade-off between problem-solving practice and worked example study in collaborative programming activities offered at scale in advanced courses teaching work-relevant computer science skills. We find results pointing to the positive impact of worked example discussions on conceptual learning from a task, thereby challenging the dominant place of problem-solving practice in the advanced computer science curriculum. Further, the study found no difference in the impact of worked example discussions and problem-solving practice on authentic hands-on task performance. A nuanced view of the separation of programming knowledge into its procedural and conceptual aspects was put forth and actualized in a collaborative learning environment. Further research is needed to confirm the results obtained, especially with respect to the lack of differential impact on authentic task performance, but evidence has been obtained to start to recommend a stronger integration of worked example study into the advanced computer science curriculum.

The literature on example-based learning points also to the effects of sequencing worked example study and problem-solving practice. Improved impact has been found by using worked example study first, to help students learn the relevant problem states, and then implement them during problem-solving practice. The study presented in this paper investigated only problem-solving practice followed by worked example study due to a lack of statistical power. This comparison has been planned for the next study in this sequence. The design

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

informed by the results presented in this paper are expected to be carried forward in forthcoming offerings of this course in community college and corporate learning contexts.

ACKNOWLEDGMENT

The authors would especially like to thank the course staff, teaching assistants, and students for steering and working enthusiastically through the course amid the COVID-19 pandemic.

REFERENCES

[1] A. Renkl, "Toward an instructionally oriented theory of example-based learning," *Cognitive Science*, vol. 38, no. 1, pp. 1–37, 2013.
[2] A. Renkl, "Learning from worked-out examples: A study on individual differences," *Cognitive Science*, vol. 21, no. 1, pp. 1–29, 1997.
[3] L. E. Margulieux, B. B. Morrison, and A. Decker, "Design and pilot testing of subgoal labeled worked examples for five core concepts in CS1," *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 2019.
[4] G. A. Dafoulas, K. Swigger, R. Brazile, F. Alpaslan, V. L. Cabrera, and F. Serçe, "Global teams: Futuristic models of collaborative work for today's software development industry," *2009 42nd Hawaii International Conference on System Sciences*, 2009.
[5] S. Sankaranarayanan, S. R. Kandimalla, C. Bogart, R. C. Murray, M. Hilton, M. Sakr, and C. Rosé, "Combining collaborative reflection based on worked-out examples with problem-solving practice: Designing collaborative programming projects for learning at scale," *Proceedings of the Eighth ACM Conference on Learning @ Scale*, 2021.
[6] S. Sankaranarayanan, S. R. Kandimalla, M. Cao, I. Maronna, H. An, C. Bogart, R. C. Murray, M. Hilton, M. Sakr, and C. Penstein Rosé, "Designing for learning during collaborative projects online: Tools and takeaways," *Information and Learning Sciences*, vol. 121, no. 7/8, pp. 569–577, 2020.
[7] S. Sankaranarayanan, "Online mob programming," *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019.
[8] M. Hilton and S. Sankaranarayanan, "Online mob programming," *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, 2019.
[9] S. Davies, J. A. Polack-Wahl, and K. Anewalt, "A snapshot of current practices in teaching the introductory programming sequence," *Proceedings of the 42nd ACM technical symposium on Computer science education - SIGCSE '11*, 2011.
[10] E. Soloway, "Learning to program = learning to construct mechanisms and explanations," *Communications of the ACM*, vol. 29, no. 9, pp. 850–858, 1986.
[11] E. Kant and A. Newell, "Problem solving techniques for the design of algorithms," *Information Processing & Management*, vol. 20, no. 1-2, pp. 97–118, 1984.
[12] E. Soloway and K. Ehrlich, "Empirical studies of Programming Knowledge," *IEEE Transactions on Software Engineering*, vol. SE-10, no. 5, pp. 595–609, 1984.
[13] J. Sweller, J. J. van Merrienboer, and F. G. Paas, *Educational Psychology Review*, vol. 10, no. 3, pp. 251–296, 1998.
[14] M. T. Chi, P. J. Feltovich, and R. Glaser, "Categorization and representation of physics problems by experts and novices*," *Cognitive Science*, vol. 5, no. 2, pp. 121–152, 1981.
[15] J. J. G. Van Merriënboer and F. G. W. C. Paas, "Automation and schema acquisition in Learning Elementary Computer Programming: Implications for the design of Practice," *Computers in Human Behavior*, vol. 6, no. 3, pp. 273–289, 1990.
[16] B. J. Ericson, L. E. Margulieux, and J. Rick, "Solving parsons problems versus fixing and writing code," *Proceedings of the 17th Koli Calling International Conference on Computing Education Research*, 2017.
[17] C. Malliarakis, M. Satratzemi, and S. Xinogalos, "CMX: The effects of an educational MMORPG on learning and Teaching Computer Programming," *IEEE Transactions on Learning Technologies*, vol. 10, no. 2, pp. 219–235, 2017.
[18] L. E. Margulieux, B. B. Morrison, and A. Decker, "Design and pilot testing of subgoal labeled worked examples for five core concepts in CS1," *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*, 2019.
[19] S. Kalyuga, P. Chandler, J. Tuovinen, and J. Sweller, "When problem solving is superior to studying worked examples.," *Journal of Educational Psychology*, vol. 93, no. 3, pp. 579–588, 2001.
[20] M. T. Chi, M. Roy, and R. G. Hausmann, "Observing tutorial dialogues collaboratively: Insights about human tutoring effectiveness from vicarious learning," *Cognitive Science*, vol. 32, no. 2, pp. 301–341, 2008.
[21] J. E. Tuovinen and J. Sweller, "A comparison of cognitive load associated with Discovery Learning and worked examples.," *Journal of Educational Psychology*, vol. 91, no. 2, pp. 334–341, 1999.
[22] P. G. Sidney, S. Hattikudur, and M. W. Alibali, "How do contrasting cases and self-explanation promote learning? evidence from Fraction Division," *Learning and Instruction*, vol. 40, pp. 29–38, 2015.
[23] A. King, "Ask to think-tel why: A model of transactive peer tutoring for Scaffolding Higher Level Complex Learning," *Educational Psychologist*, vol. 32, no. 4, pp. 221–235, 1997.
[24] G. Gweon, C. Rose, R. Carey, and Z. Zaiss, "Providing support for adaptive scripting in an on-line collaborative learning environment," *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2006.
[25] A. Wilson, "Mob programming - what works, what doesn't," *Lecture Notes in Business Information Processing*, pp. 319–325, 2015.
[26] W. Zuill and Kevin Meadows, "Mob programming: A whole team approach," *Agile 2014 Conference, Orlando, Florida*, vol. 3, 2016.
[27] W. C. Newstetter and C. E. Hmelo, "Distributing Cognition or How They Don't: An Investigation of Student Collaborative Learning," *Proceedings of the 1996 international conference on Learning sciences*, pp. 462–467, 1996.
[28] S. M. Carver, "Cognition and instruction: Enriching the laboratory school experience of children, teachers, parents, and undergraduates," in *Cognition and instruction.: Twenty-five years of progress*, Mahwah, NJ: Lawrence Erlbaum Associates, 2001.
[29] S. Sankaranarayanan, S. R. Kandimalla, M. Cao, I. Maronna, H. An, C. Bogart, R. C. Murray, M. Hilton, M. Sakr, and C. Penstein Rosé, "Designing for learning during collaborative projects online: Tools and takeaways," *Information and Learning Sciences*, vol. 121, no. 7/8, pp. 569–577, 2020.

Sreecharan ``Sree" Sankaranarayanan is a Ph.D. candidate (in absentia) at the Language Technologies Institute of Carnegie Mellon University. His research focuses on the use of conversational agents to support group-work in online learning contexts. As a part of the Technology for Effective and Efficient Learning[3] Lab, his work has been applied to project courses offered to several hundred students each semester at Carnegie Mellon University and its worldwide branch campuses. As the co-director of Discussion Affordances for Natural Collaborative Exchange[4], his prior work, in partnership with edX, brought conversational agent-based support to several thousand students online. He served, formerly, as the Founding Co-Chair of the International Learning Sciences Student Association (ILSSA), a committee of the International Society of the Learning Sciences (ISLS) for students of the learning sciences. He is currently a Research Scientist at Amazon continuing his work in the workplace learning context.

[3] http://teel.cs.cmu.edu/

[4] http://dance.cs.cmu.edu

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

Siddharth Reddy Kandimalla is a project scientist for the Technology for Effective and Efficient Learning[3] Lab at the Computer Science Department of Carnegie Mellon University. Siddharth leads the design and development of the Social and Interactive Learning (SAIL) platform jointly under development by Carnegie Mellon and Microsoft Azure focused on post-secondary education in the field of cloud computing, data science, machine learning. He leads the software engineering team that manages 35 cloud-native microservices of the SAIL platform, currently used by multiple courses at Carnegie Mellon University and tens of external universities, serving thousands of requests per day. He received his Master's degree in Information Systems & Management from Carnegie Mellon University.

Dr. Christopher A. Bogart is a system scientist at the Institute for Software Research at Carnegie Mellon University and a leadership team member of the Technology for Effective and Efficient Learning[3] Lab. His research agenda focuses on studying coordination, allocation of effort, diffusion of practices, and skill learning among software developers, with the goal of improving the sustainability of open-source infrastructure and growing and diversifying the workforce of CS and IT professionals. His recent research threads include studying the extent to which open-source ecosystems' chosen practices in fact depend on their values, history, context, and mix of stakeholders; and helping bring new computer science and IT professionals into the field by improving the effectiveness of scalable online project-based learning techniques. He holds B.S., and M.S. degrees in Computer Science from Colorado State University and a Ph.D. from Oregon State University.

Dr. R. Charles Murray is a research developer at the Language Technologies Institute at Carnegie Mellon University and lead developer of the Bazaar architecture for collaborative conversational agents. Since earning his Ph.D. in Intelligent Systems at the University of Pittsburgh, his focus has been on developing systems that use artificial intelligence to engage and educate users. As a Cognitive Scientist for Carnegie Learning, he designed and developed a large proportion of the company's Cognitive Tutors, which have been used by millions of students and are widely recognized for delivering effective instruction. At Carnegie Mellon University, he has moved on to developing systems for effective collaboration between people and between people and computers. His recent work extends to multimodal interaction which takes as input users' spoken language, facial expressions, physical location, and body position, and responds with spoken dialog along with appropriate facial expressions, gaze direction, and body gestures.

Michael Hilton is an Assistant Teaching Professor of Computer Science at the Institute for Software Research of the School of Computer Science at Carnegie Mellon University. He the director of the undergraduate software engineering minor as well as the Software Engineering Concentration at the School. His teaching explores the implications of startups on the practice of software engineering. His research focuses on understanding developers and development processes. In particular, his research focuses on Continuous Integration, understanding its use, and seeking ways to find improvements for developers. He received his B.S. from San Diego State University, his M.S. from Cal Poly San Luis Obispo, and his Ph.D. from Oregon State University, all in Computer Science.

Majd F. Sakr is is a Teaching Professor in the Computer Science Department within the School of Computer Science at Carnegie Mellon University. He is also the co-director of the Master of Computational Data Science (MCDS) Program at the School of Computer Science at Carnegie Mellon. From 2007-2013, Majd was a Computer Science faculty at Carnegie Mellon University in Qatar (CMUQ), where he also held the positions of Assistant Dean for Research and Coordinator of the Computer Science Program. He founded the Cloud Computing Lab and co-founded the Qri8 Qatar Robotics Innovation Lab at CMUQ. He also co-founded the Qatar Cloud Computing Center. He has held appointments at the American University of Science and Technology in Beirut and at the NEC Research Institute in Princeton, New Jersey. His research interests include online education, cloud computing, and human-robot interaction. He holds a B.S., M.S. and Ph.D. in electrical engineering from the University of Pittsburgh.

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TLT.2022.3169121, IEEE Transactions on Learning Technologies

Carolyn P. Rosé is a Professor of Language Technologies and Human-Computer Interaction in the School of Computer Science at Carnegie Mellon University. Her research program focuses on computational modeling of discourse to enable scientific understanding of the social and pragmatic nature of conversational interaction of all forms and using this understanding to build intelligent computational systems for improving collaborative interactions. Her research group's highly interdisciplinary work, published in over 270 peer reviewed publications, is represented in the top venues of 5 fields: namely, Language Technologies, Learning Sciences, Cognitive Science, Educational Technology, and Human-Computer Interaction, with awards in 3 of these fields. She is a Past President and Inaugural Fellow of the International Society of the Learning Sciences, Senior Member of IEEE, Founding Chair of the International Alliance to Advance Learning in the Digital Era, and Co-Editor-in-Chief of the International Journal of Computer-Supported Collaborative Learning. She also serves as a 2020-2021 AAAS Fellow under the Leshner Institute for Public Engagement with Science, with a focus on public engagement with Artificial Intelligence.

2372-0050 (c) 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See http://www.ieee.org/publications_standards/publications/rights/index.html for more information.
Authorized licensed use limited to: Carnegie Mellon Libraries. Downloaded on September 19,2022 at 18:29:13 UTC from IEEE Xplore. Restrictions apply.