The Programmers' Collective: Connecting Collaboration and Computation in a High School Scratch Mashup Coding Workshop

Deborah Fields, Utah State University, 2830 Old Main Hill, Logan, UT
Veena Vasudevan and Yasmin B. Kafai, University of Pennsylvania, 3700 Walnut Street, Philadelphia, PA
deborah.fields@usu.edu, veenav@gse.upenn.edu, kafai@upenn.edu

Abstract: One challenge in bringing collaborative programming into schools has been to develop software design activities that make computation an integral part of the collective design rather than just a work arrangement between team members. In this paper we propose a collaborative approach to programming, the programmers' collective, that builds on collaborative models found in do-it-yourself (DIY) and open source communities. We describe and analyze the work of a class of high school student collectives engaged in programming music video mashups as part of a collaborative challenge in the online Scratch community. Our multi-level analysis focused on students' learning of specific programming concepts, effects of collaborative and task design on learning, and their personal reflections. In the discussion we address how students' experiences point to the value of "nested collectives," or multiple levels of designed-for collaboration as well as a needed shift from a focus on computation to computational participation.

Introduction

In computer science education, engaging and supporting students in collaborative programming has been a central focus of research for multiple reasons: the educational benefits that collaboration has shown to offer in learning interactions, the collaborative nature of software design and related industry, and resource allocation that allows students to share a limited number of computers in classrooms. Yet many early studies have not found collaborative programming efforts in K-12 to be successful in helping all members of teams develop programming skills (e.g., Webb, Ender, & Lewis, 1986). More recently pair programming approaches, in which two students work together on programming tasks, have been favored. Research has shown that pair programming can be a very effective collaborative arrangement for both K-12 and college students in helping them learn key computational concepts and persisting in programming (Denner & Werner, 2007; Porter, Guzdial, McDowell, & Simon, 2013). Yet while pair programming may meet some of the learning goals and resource needs of schools, it fails expose students to the collaboration so commonly reflected in professional software design and self-organized Do-It-Yourself (DIY) online creative collaborations.

Most pair programming efforts that take place in schools often assign students to teams and provide them with specific tasks to be accomplished. This is substantially different from collaborative programming in online communities which focus more on self-selected tasks and work arrangements. Unlike classrooms, in this latter collective style of programming, members might not have met in person and are often geographically distributed, working across different time zones. Most prominent examples of open-source community design, such as the design of the operating system Linux (Benkler, 2006) or computer animations in Newgrounds (Luther, Caine, Ziegler & Bruckman, 2010) are facilitated by adult members. These types of collaborative activities have also found their way into youth programming communities such as Scratch where large numbers of youth can often be found collaborating together on programming role playing stories, animations, and games (Resnick et al, 2009). Integrating this type of collaborative programming into school poses a challenge (Kafai et al., 2012) because the DIY mentality of free forming groups and flexible work arrangements does not fit well within the more structured organization of schools.

In this paper, we report on the implementation and analysis of a collaborative, computational design challenge where students worked in collectives to create music video mashups in Scratch. This pilot study focused on structuring collaboration through a programming task so we could make this type of DIY collective programming available in a school setting. Conscious of tendencies for groups to divvy up tasks and reify stereotypes by assigning harder tasks to people who are perceived to be better at computers (Dumont & Fields, 2013; Kafai et al., 2012), we structured the programming task such that each individual worked on a section of their group's song. Then each group integrated the sections together into one longer music video, situating the learning of key computational concepts of initialization, synchronization, and event-driven programming. Our research questions asked: In which ways does a collective programming task support the learning of key programming concepts, especially of initialization and synchronization? What do students have to say about their collective programming experience? In the discussion we address what we learned about students' collaborative programming experiences, the design of collective programming tasks, and the integration of this type of collective programming activity in school contexts.

Background
There is no question that learning to program is challenging. As research over the last 30 years has documented in hundreds of studies in schools, students encounter challenges with both syntax and control and data structures (e.g., Robbins, Rountree, & Rountree, 2003; Soloway & Spohrer, 1990). Much of the recent comeback of programming has taken place outside of schools in online communities in which youth engage in the production of games, animations, role-playing games, and worlds (Kafai & Burke, in press). In these interest-driven communities youth come together not only to hang out and mess around (Ito et al., 2010) but also to create, remix, and share their code. While gaining access to and participation in programming communities is not that easy (Fields, Kafai & Burke, 2009), the work arrangements in online communities are even harder to replicate in school settings that have more structured time and less flexibility, leaving students ill-prepared for managing these more open-ended collectives found online.

In designing the programmers' collective, we built on online, interest-driven collaboration observed "in the wild," but intentionally developed supports for collaborations. Using an iterative design research approach, we designed and implemented collaborative programming tasks for all members in the Scratch online community to join. In successive iterative implementations we moved from a Collab Challenge in which online groups were asked to program three objects (Kafai, et al., 2012), to a Collab Camp with a more story-oriented format that asked groups to write and program an interactive story (Roque, Kafai & Fields, 2012). Our analyses of how online groups came together (or not) to accomplish these tasks led us to formulate a need for students to develop "collaborative agency"—a concept that draws on Scardamalia's (2002) work on collective cognitive responsibility—meaning that learners have to assume agency in finding others, organizing work, and contributing to and critiquing designs (Kafai et al., 2012). By all accounts these are desirable skills for students to develop in collaborative contexts and that can be supported via design features such as providing spaces to find collaborators and share curated work in galleries, furnishing mechanisms for attribution (Monroy-Hernandez, 2012), and modeling constructive criticism. These findings pointed to the need to design for what we call *nested collectives*—multiple levels of collaboration that attend to individual learning in the context of different layers of collaborative support.

In this third iteration of collaborative projects, Collab Camp 2, we focused on supporting individual learning through participation in a collective programming task. The design of the task would ensure that students encounter two core concepts that most novice programmers find challenging: initialization and synchronization. Initialization is an important concept in computer programming because it suggests a starting state or point for all of the actions and actors in a program, important enough that many modern languages provide means for automatically enforcing or checking initial states of variables. In Scratch, initialization becomes an intentional design because initial states of where something should start need to be clearly defined. When students run their programs repeatedly, they begin to notice that the program is not behaving as they desire. For example, a programmer may command a sprite (a character or image that is directed by commands) to move across the screen. On subsequent runs of the program, the sprite will begin in the last place it ended during the prior run, demonstrating the importance of initializing a sprite's x and y coordinate location at the start of the program. Thus, programmers need to learn to set a sprite's appearance, state (e.g., showing or hidden), size, and direction. A second goal was to support learning of synchronization and event-driven programming. Synchronization and event driven programming involve coordinating processes and events within a program such that they can be executed in parallel or sequentially without bringing the program to a stand still. In Scratch, coordinating multiple sprites' actions involves learning different forms of synchronization and event driven programming (Maloney et al., 2008; Fadjo, 2012).

For this study, we situated our programming task in an authentic form of collective youth media production: making (programming) video mashups of popular music. In order for a music video to flow smoothly, all sprites' locations, appearances, and sizes from each segment needed to be initialized and each event (i.e., each segment) needed to be triggered through synchronization and event commands. While this design of the programming task focused the coordination of individual contributions by making them essential to the completion of the project, we also recognize the role that larger communities, offline and online, played in supporting and providing feedback. Each programmers' collective became part of increasingly larger collectives—smaller groups within a larger workshop group within a broad, online challenge that all worked on programming music mashups. Our analysis focuses primarily on the design task but also attends to the supports for learning and collaboration at the larger collective levels as well.

Participants, Context, and Methods
This research was conducted at a high school situated in a metropolitan city in the northeastern United States. This high school is known for their intensive focus on science, technology, engineering and math (STEM). Freshman students (ages 14-15) participated in this workshop as part of their enrichment curricula in partnership with a local museum. Twenty-nine high school students participated in the workshop (12 girls, 17 boys) and of those, 23 (9 girls, 14 boys) consented to participating in the research. This included five male "mentors" in their

sophomore year that had participated in Scratch workshops the year prior. A team of researchers from the University of Pennsylvania taught the workshop: one main instructor led the course and was supported by two additional graduate student researchers. Five undergraduate interns also periodically supported students.

The students in the workshop participated in an online design challenge hosted on Scratch.mit.edu: completing a video music mashup as part of a collaborative team. The online challenge, Collab Camp 2 (see Roque, Kafai & Fields, 2012), had minimal requirements: participating groups had to submit a draft and finished version of their mashups and groups had to be comprised of two or more people. In our workshop we added a few additional constraints to support students' collaboration and learning: 1) students worked in collectives of 3-6 (versus the minimal requirement of two members), 2) each collective made a music video (versus the more general "music mashup" online), 3) each member had to design a section of the song their group chose (versus no structure on how to collaborate). The workshop comprised of eight sessions that ran for two hours each week in Winter 2012. In the early sessions, students selected songs they wanted to mash up, and were sorted into groups based on song choice. Then, each student selected a section of the song to animate or illustrate in Scratch. In addition, during the early weeks, the collectives were given a short programming tutorial and a few lines of starter code that used a particular mechanism for synchronizing events with time in the music video. During the sixth session groups merged their segments together to submit one seamless music video draft to the online challenge. Alongside other online-based groups, the collectives received external feedback from experts (youth and adult) on the Scratch website by the seventh session, and proceeded to revise and finesse their projects for the final submission to the website and presentation in the workshop.

Data from the project include weekly interim and final Scratch programs, observational data including field notes and video logs, and post-hoc interviews with a subset (11 of 25) of the students including three student mentors. To analyze student learning of specific programming concepts we turned to their programs, analyzing their music mashups for evidence of initialization and synchronization. We developed a coding scheme to determine the frequency and range of the initialization and synchronization strategies as well as how consistently teams employed these across individual sections. This analysis helped us to see how teams collaborated to ensure flow in their music videos. To further enrich this analysis, we used the interviews to understand students' perceptions towards collaboration using a two-step open-coding scheme (Charmaz, 2000). We describe the key emergent themes—collaboration, engagement/motivation, and computation and learning—in more detail in the last sections of findings.

Findings

Our findings illustrate how the collaborative programming task was successful in getting all individual students and the collectives overall to employ a range of initialization and synchronization strategies to produce cohesive music videos. We also show how the nested nature of the collectives influenced students' designs and opportunities for learning and feedback. First, we will describe the key initialization and synchronization strategies used. Then, we present a case study to illustrate one collective's collaborative process of how the workshop design, team structure, design task and informal and formal feedback helped shape their final music video, which was emblematic of other students' experiences. Finally, we consider students' reflections on nested collectives, including their learning from high school mentors, college interns, and members of the broader online challenge.

Initializing and Synchronizing Music Segments in Programmers' Collectives

To manage the programming task, each student had to think about how to most meaningfully bring their sections of music or lyrics to life. Given that their code would eventually be merged with that of members of their collective, they also had to be cognizant of initializing appearances, actions, and variables as well as synchronizing characters' actions and movements so the music video flowed. However, being new to Scratch, this required students to navigate a significant series of challenges from being creative about how to bring their lyrics to life to knowing how to develop the code to make their intended actions happen. We identified two programming concepts as targets for learning: initialization and synchronization & events. In our analysis of the final programs, we found that all individual students eventually employed initialization systematically throughout their projects. Programming an initialization involved setting the beginning state of sprites' (Scratch characters') appearance (e.g. *hide, show*), location (e.g. *go to x=__, y= __*), costume (e.g. *switch to costume__*), direction (e.g. *point in direction*), graphic effects (e.g. *set size to__*), and layering (e.g. *go to front*). If variables were used in a project, students consistently initialized the variable values as well. In early versions of the projects, initialization strategies were not used consistently, yet by the time collectives submitted their first draft of projects most sprites were initialized in the ways detailed above. Observations showed that as students combined their segments together any gaps in initialization were revealed since sprites from a later segment would appear at the beginning of the song or in the wrong place on the screen (for more detail, see the case study of the Jane Doe collective). This suggests that the design of the collaborative programming task of bringing individual segments together into a chronological, unified project was successful in supporting students' systematic learning of this computational concept.

In terms of synchronization, individual students applied a variety of synchronization strategies including those suggested in the starter code given to each group. The most basic strategies included using *when green flag clicked* and *wait* to synchronize events. Students used *when green flag clicked* to start the main sequence of the music video. The more sophisticated scripts developed by students made use of the scripts *wait until* and *broadcast* to initiate as well as to conclude a sequence. As prompted by the starter code given to each group, students most often used *wait until timer > X*, to start their sequences. All members of a group had to coordinate their start and end times with the global music video timer, thus, this command became essential in setting boundaries for individuals' sections within the music video. Demonstrating that they understood the utility of this command, many students also used it to end their sequences, for instance continuing to animate a sprite(s) until the timer reached a certain time and thus having the active sprites disappear from the stage to make way for the next sequence. The seamless transitions throughout music videos indicated that the collaborative design task was also successful in getting students to consistently use synchronization strategies.

Students employed a variety of strategies to ensure that their precisely timed video segments flowed together smoothly. Of course, some students' code was more sophisticated than others. Students brand new to Scratch had simpler code and visual effects than those who had spent one prior workshop in Scratch, especially those five sophomore mentors who had participated in more than four prior workshops in Scratch. Yet even the more experienced sophomore mentors expressed that they learned something new through this design challenge. Some had not used the "wait until" command before and those that had realized that they had not used it effectively. As sophomore mentor Jacob pointed out, in earlier projects he had synchronized everything with *wait* commands, but this strategy was "a lot more messy and took a lot more time" than using the more advanced *wait until* command. This shows an advancement in use of synchronization strategies. Thus the music mashup challenge promoted learning at multiple levels, from programming novices to more advanced students. This raises the question of *how* students learned initialization and synchronization strategies within their small collectives' group negotiations? To answer this question, we turn to a case study of one programmer collective: the Jane Doe group.

Case Study of Jane Doe Programmer Collective

The Jane Doe group's collaboration is emblematic of how the design task and collaborative team structure (e.g. the programmers' collective) together led to a an intricate and well-produced final music video, with the group relying on each other as well as the nested collectives of the larger workshop group and online community to finish their program. Comprised of five girls and one boy—Marjorie, Leslie, Aris, Bridget, Natalie, and Oswald—the Jane Doe group chose to make their music video mashup to the song "Jane Doe" by *Never Shout Never*. The song describes a boy falling in love at first sight with a girl whose name he doesn't know. Joined by a common interest in the song, the students formed the group and divided the song into seven main parts with one member in charge of each segment (see Figure 1), and one additional segment (a musical interlude) unassigned at the beginning.

In their early sessions, the collective faced the challenge of creating separate roles as well as an underlying aesthetic theme for the group. They were unsure of how to effectively parse a song less than two minutes long while maintaining consistency in terms of the aesthetic design across their individual contributions. The group eventually rallied around the idea of dividing the song into six short segments based on natural breaks in the lyrics and giving each section its own unique look and feel while keeping the theme of an unidentified girl (a "Jane Doe") throughout. Early on in the process, one of the students, Oswald, asked for the shortest part of the song. In exchange he agreed to consolidate all of the sections when the time came. The students proceeded to program their own sections with regular feedback from each other until the deadline for the first submission online when Oswald led the integration of sections. This became a key moment in their developing understanding of initialization and synchronization.

The most demanding programming happened when the group first merged all of their programs into one music video mashup. In her reflection, Leslie acknowledged that this was most challenging part since they discovered that suddenly all the sprites from each segment showed up on the screen at the same time. She explained that the team resolved this by initializing each sprite to *hide* and only *show* when the timing was appropriate, explaining, "Um, so we hid all the sprites and we used a timer to make sure everything comes up at the right time." As predicted, merging all of the segmented programs brought the issue of initialization to the fore. By the end of the project, each student in the Jane Doe group consistently initialized all of their variables in every relevant aspect including location, layering, and appearance so that the final sections flowed smoothly and seamlessly.

The group also faced some organizational programming challenges, especially in regard to the synchronization of the project. In keeping track of who was doing what, early on the group employed some basic organizational strategies like allocating specific timings, maintaining a sheet that held those timings, and giving everyone the space to be creative by working on their individual parts. Eventually those written down timings came in handy as the group centralized the main synchronization and control of the program in the "stage," a centralized programming strategy that helped eliminate the confusion of code spread more diffusely. Within each individual program the biggest sections were synchronized with the *wait until [timer > X]* command, and within those blocks of code students used *wait X seconds* or *broadcasts* to control smaller actions. That the group worked well together, despite the challenges they faced, is evidenced in that this collective was the first to finish their coding mashup and managed to merge their program into one final music video without the support of teachers or mentors.

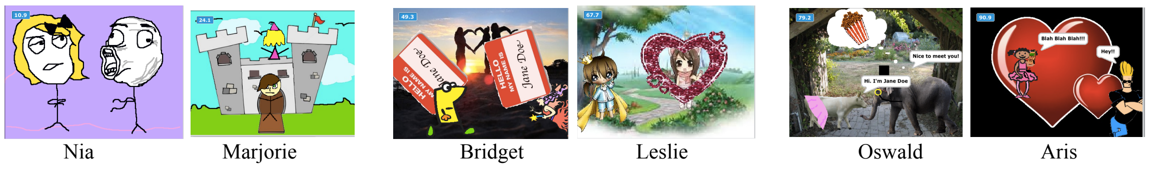

Figure 1. Screenshots from each student's scene in Jane Doe

Interestingly, through the conversations that the group had over the course of the workshop as well as their final reflective interviews, it is evident that being in a group shaped their decision-making as well as their sense of contribution and learning. Students stated that working in a group helped them to learn more. As Marjorie expressed, "I guess I learned more from my group mates and like in the beginning I felt I didn't know that much, like the first workshop. Then I was with my group and I sort of learned more than I did the first time." Group members also got ideas from each other, and not only from each other but from the larger collective of the workshop group as a whole. As an illustration, two members who were also friends, Marjorie and Leslie, opted to work together on the musical interlude in the song because they had finished their individual segments early. They developed the concept and the code together, with Leslie explaining that in this collaboration her partner "thought of the ukulele switching back and forth and I thought to have the music notes like change colors and bounce." Taking turns at the computer they together programmed the interlude with repeating patterns that were unique to other segments of the mashup. Oswald's section of the mashup further reveals the role the larger collective of the workshop played in the group's design. Initially Oswald was not sure what to include in his music video. In the third week of the course, he was cajoled by his group to put some action into his segments and then one of the undergrad interns helped him brainstorm, eventually giving the idea to use animals as his main sprites. In week five, one of the teachers asked Oswald to demo his section of the music mashup, and afterwards students were asked to give him feedback. During this time, students from the larger workshop and one of the teachers shared different ideas related to imagery and additional action that Oswald could include. He eventually opted to incorporate a combination of these ideas into his final segment. (Notice the popcorn thought bubble in Figure 1, an idea generated from the workshop feedback session). In Oswald's case, his immediate group members, the undergraduate mentors, and the larger workshop gave him feedback that helped to shape his and others' final products. This points to the role of nested collectives in the group's success: the smaller collective of Jane Doe's individual team embedded in the larger collective of the workshop which provided external feedback, ideas, and support. Below we turn to students' reflections on computation and collaboration across the whole workshop.

Perspectives on Collaboration, Support, and Programming

Students' reflections captured in interviews at the end of project provide an additional perspective on computing and collaboration that centered on three aspects: (1) value of collaboration, (2) presence of mentors, and (3) attitudes towards computing. Students articulated that collaboration was central to their learning and strongly influenced their final projects. They explained that even while there were challenges to collaborating in groups there were some general benefits as well. First, they expressed ownership in the entire project, realizing that their individual sections were part of something bigger that they could not have completed independently. For example, Ashton mentioned that while he preferred individual work in general, he could not have accomplished the big music video task without the team: "Personally, I don't like working in groups because if I do it all by myself, I know it's going to get done… But in that kind of project, in eight weeks… if you can't finish [your] part, how are you going to finish a four minute music video?" A similar sentiment was also expressed by Tighe who said, "You can't just do it all by yourself and...you got to depend on your team and work together to do it because eight weeks... eight classes... It's just hard." These types of comments point to the authenticity of the programming task: collaboration was necessary in order to accomplish it, and students expressed pride in the final products, which were bigger than any individual could have completed alone.

In addition to acknowledging the importance of collaboration for the complexity and size of the task, students also expressed that they learned and got ideas from their group members. For example, Aris stated that working in groups was better than working individually "because you have more ideas." Similarly, Justus shared that he found that his group helped keep him focused instead of “lagging off” and made his ideas better: “When I worked individually, I kind of lagged off a little bit. You know, it's just me. Also, it [the group] helped me come up with the ideas faster. But all-in-all, I had a better idea in the group.” Students also explained that an advantage to working in groups was being able to learn from their peers. For example, Belle explained that working in a group is better than working individually because “Working in a group, you learn new things that you haven't learned before that you can't really learn by yourself. It's like, ‘How do I do this?’ and then you ask the person next to you and he'll say, "Well, here's an easier way." And you're like, ‘Oh, I never knew that before.’ The high school mentors further explained this collaborative advantage, with one mentor pointing to the fluid group dynamics as the reason for his group’s success. In another case, the mentor explained that the group had a student who wasn’t as comfortable in Scratch and that the other group members stepped into support her, stating, “Yeah, so we all kind of were able to work as a team to help the struggling members of the team.” He also argued that through collaboration and mutual support, the entire team was able to be successful.

The presence of mentors, interns, and teachers helped students bring their ideas to life in Scratch and in the process, broaden their understanding of the different approaches to solving problems. Mentors were seen as a meaningful source of support in both learning how to navigate programming challenges and in staying on track. Students reflected that the mentors’ knowledge of Scratch provided them insights into how to approach different challenges. Belle said of her mentor, “during our making our videos, I asked my mentor a lot of questions and he taught me to use different things [in Scratch],” indicating that the mentors could help youth work out challenges through active problem solving with them. Another student Justus, explained that in one instance, his mentor helped him strategize what synchronization commands he should use for a particular part of his project using this interactive approach. In addition to helping solve problems, several students indicated that mentors were there to keep students on track or help them out when they encountered challenges such as getting stuck. Stephen, for example, explained how the mentors’ role was “pushing everybody to do their parts.” Another student, Charlotte, explained that the college intern helped her group by checking in on them: “She would check up constantly. Like, ‘hey, where are you? You know you have to be finished by a certain time and upload,’ so that was actually very helpful.” Justus, who worked with this same college intern explained that when he got distracted, “Tara [intern] was always there to reel me back into the project. She did a good job of keeping us all working on it.” These are helpful illustrations of the kind of informal and critical support that mentors, interns and teachers provided that furthered opportunities for nested collaboration in the classroom (see also Kafai, et al, 2013).

Finally, all students expressed how their understanding of programming shifted favorably through their participation in the music mashup challenge. The students’ reflections illustrated how extended time in the workshop setting gave them opportunities to practice and overcome initial frustrations or in some cases skepticism or disinterest. For example, George said, “...when you first start Scratch, it's kind of like this is really hard, I don't really want to do this... But like once you start getting... knowing things it's really fun.” George expressed the change that occurs when there is time to practice. Another student, Aris, explained that her understanding of programming changed stating, “it’s not as hard as I thought it was” and noting that as an eighth grade student the year prior she would not have seen herself enjoying a workshop like Scratch. Both George and Aris’ reflections demonstrate how participation helped them to see they could be successful and that Scratch was not as hard as they initially imagined, challenging their preconceived notions about programming. Other students pointed to the possibilities available through the Scratch platform as a compelling reason to participate. For example, Stephen explained that he thought, “Scratch is very fun, because you can actually see your creative thinking to do anything you want as long as you can... like you have to work out the bugs and problems. But if you can do that, then there's almost endless possibilities.” These same sentiments were evident in the mentor interviews. Two high school mentors argued that they were impressed with the growth they saw amongst students they worked with during the workshop. They saw students go from not knowing or understanding how to navigate the space to being much more experienced and comfortable programmers of Scratch.

Discussion

Our goal in examining the programmers’ collective was to understand better how we can bring the DIY dimensions of dispersed online creative communities to face-to-face workshops and classrooms introducing programming to high school students. There are, of course, other models of such learning communities such as the software design studio (Kafai & Burke, in press) and Knowledge Forum (Scardamalia, 2002) that embody many of the same ideas. We chose as a starting point a software design task that not only encouraged more complex programming for beginners but also invited better coordination for collaboration. Our design parameters for collaborative programming involved choosing a relevant task such as a music video mashup that fit well within popular youth media production and drove student interest in a culturally relevant manner (Enyedy, Danish & Fields, 2011). Furthermore, what seemingly looked like a trivial task became a rich programming activity when turned it into a collective programming design. Two key computational concepts, initialization and synchronization, were needed to connect different parts of the video into a smooth production.

Thus learning and implementing these concepts were instrumental to accomplishing the collective design. As such the programming project had an interdependent design structure that required individual contributions to accomplish a task larger (and more desirable) than what one could do on one's own. Motivating learners in design activities has been a longtime challenge (Collins, Joseph, & Bielaczyc, & Josephs, 2004) and relates to the current focus on interest-driven and connected learning (Ito et al., 2013).

The need for developing and researching the design of such collaborative programming tasks is crucial because pair and group programming activities have met with varying success. Practically, there are not many ways for two people to concurrently program, which is not the case with other activities like writing which can be a concurrent and collaborative activity with applications like Google Drive. However, this study helped us to think more creatively about the design task and our goal with this work. Could we have done more to ensure higher levels of group collaboration? Or was the design task collaborative enough? On the surface, it would seem that the groups only had to collaborate at the end to coordinate their projects. This approach was one effort to ensure individual learning and ownership of a product while also promoting learning from each other through creation of a shared product within a group and through regular viewing of others' work throughout the workshop as well as videos online. As we saw in the findings, the design/structure of the physical space (e.g., kids seated with their groups) and assigning mentors and interns to groups, coupled with the opportunities for informal and formal feedback at multiple levels did in fact lead to high levels of collaboration. Throughout our observations, we found that simply sitting together allowed youth to ask questions to their teammates. In addition the structure of the class, where there was a lot of additional support available, allowed kids to request support but also receive help when mentors, interns and teachers informally circulated the room to check-in with groups. Students also had multiple sources for formal feedback, both online and offline (e.g., during group shares), actually resulting in changes to their final products. So, by designing a collaborative task and also creating a collaborative space, the workshop encouraged different kinds of collaboration. Evidence suggests that we were successful in promoting individual learning, ownership, and engagement in the programming task while also promoting a sense of learning from each other and collective ownership of a final programming project.

Beyond the classroom, the ability to learn to work with others in today's highly distributed, networked world is paramount. What we called collaborative agency in an earlier paper (Kafai et al, 2012) speaks to students' ability to self-form collectives or collabs to accomplish larger tasks than they could on their own. Yet like others before us, we found that while students can often figure out how to accomplish problems, they do not necessarily equitably divide tasks and often reinforce stereotypes about who has expertise in what. Thus in this paper we addressed a key challenge of collaboration: that of equity in individual participation and contribution. Note that we stress equity and equality in collective activities, where members contribute based on their expertise and distribute responsibilities for programming to everyone, not just relying on those perceived most capable. Expanding on our earlier definition (Kafai et al, 2012), collaborative agency means that members of a team need to reach out to each other and coordinate a task equitably, precisely what the music video task required and scaffolded through its design. Finally, we suggest the idea of *nested collectives* to acknowledge the instrumental role that not just group members but others such as mentors, teachers, and students in the larger workshop played in supporting youth programming activities. Not only this, but these programming collectives were nested within the larger online challenge and students gained much from online feedback and from seeing others' projects from around the world. Like a community of learners in Lave & Wenger's sense (1991), we brought in several layers of more advanced members, including mentors from the same school with one additional year of experience, undergrad mentors from a local university, teachers, and active members of the online Scratch community.

References

Benkler, Y. (2006). *The wealth of networks: How social production transforms markets and freedom.* New Haven and London: Yale University Press.

Charmaz, K. (2000). Grounded theory: Objectivist and constructivist methods. In N. K. Denzin & Y. S. Lincoln (Eds.), *Handbook of Qualitative Research* (pp. 509-535). Thousand Oaks, CA: Sage Publications.

Collins, A., Joseph, D., & Bielaczyc, K. (2004). Design research: Theoretical and methodological issues. *The Journal of the learning sciences*, *13*(1), 15-42.

Denner, J., and L. Werner. "Computer Programming in Middle School: How Pairs Respond to Challenges." *Journal of Education Computing Research* 37, no. 2 (2007): 131-50.

DuMont, M. & Fields, D. A. (2013). Hybrid shmybrid: Using collaborative structure to understand the relationship between virtual and tangible elements of a computational craft. In N. Rummel, M. Kapur, M. Nathan, & S. Puntambekar (Eds), *To see the world and a grain of sand: Learning across levels of space, time, and scale: CSCL 2013 Conference Proceedings, Volume 2, Short Papers, Panels, Posters, Demos & Community Events*. International Society of the Learning Sciences: Madison, WI, 233-234.

Enyedy, N., Danish, J. & Fields, D. A. (2011). Negotiating the "relevant" in culturally relevant mathematics. *Canadian Journal of Mathematics,* 11(*3*), 273-291.

Fadjo, C. L. (2012). Developing Computational Thinking Through Grounded Embodied Cognition. Unpublished dissertation. Columbia University.

Ito, M., Baumer, S., Bittanti, M., boyd, d., Cody, R., Herr, B., Horst, H. A., Lange, P. G., Mahendran, D., Martinez, K., Pascoe, C. J., Perkel, D., Robinson, L., Sims, C., & Tripp, L. (2010). *Hanging out, messing around, geeking out: Living and learning with new media.* Cambridge, MA: MIT Press.

Kafai, Y. B. & Burke, W. Q. (in press). *Connected Code: Children as the Programmers, Designers and Makers for the 21st century*. Cambridge, MA: MIT Press.

Kafai, Y. B., Griffin, J., Burke, W. Q. Slattery, M., Fields, D. A., Powell, R., Grab, M., Davidson, S. B. & Sun, J. S. (2013). Broadening participation in and perceptions of computing through cascading mentoring: Implementing a CS community service learning course. In *Proceeding of the 44th ACM technical symposium on Computer science education* (SIGCSE '13). ACM, New York, NY, USA, 101-106.

Kafai, Y. B., Fields, D. A., Roque, R., Burke, W. Q., & Monroy-Hernández, A. (2012). Collaborative agency in youth online and offline creative production in Scratch. *Research and Practice in Technology Enhanced Learning*, *7*(2), 63-87.

Luther, K., Caine, K., Ziegler, K., & Bruckman, A. (2010). Why it works (when it works): Success factors in online creative collaboration. In *GROUP '10: Proceedings of the ACM Conference on Supporting Group Work* (pp. 1–10) New York: ACM Press.

Maloney, J., Peppler, K., Kafai, Y., Resnick, M., & Rusk, N. (2008). *Programming by Choice. Urban Youth Learning Programming with Scratch*. Paper presented at the SIGCSE 2008 Conference, Portland, Oregon.

Monroy-Hernandez, A. (2012). Designing for remixing: Supporting an online community of amateur creators. *Unpublished dissertation*. MIT.

Porter, L. Guzdial, M., McDowell, C. & Simon, B. 2013). Success in introductory programming: what works? *Communications of ACM, 56*(8), 34-36.

Resnick, M., Maloney, J., Hernández, A. M., Rusk, N., Eastmond, E., Brennan, K., Millner, A. D., Rosenbaum, E., Silver, J., Silverman, B., & Kafai, Y. B. (2009). Scratch: Programming for everyone. *Communications of the ACM, 52*(11), 60–67.

Robbins, A., Rountree, J., & Rountree, N. (2003). Learning and Teaching Programming: A Review and Discussion. *Computer Science Education, 13*(2), 137-72.

Roque, R., Kafai, Y. B., & Fields, D. A. (2012). From tools to communities: Designs to support online creative collaboration in Scratch. *Proceedings of IDC 2012.* Bremen, Germany.

Scardamalia, M. (2002). Collective cognitive responsibility for the advancement of knowledge. In B. Smith (Eds.), *Liberal education in a knowledge society* (pp. 67–98). Chicago: Open Court.

Soloway, E., and Spohrer, J. (1990). *Empirical studies of novice programmers.* Norwood, NJ: Ablex Publishing.

Webb, N. Ender, P. & Lewis, S. (1986). Problem solving strategies and group processes in small groups learning computer programming. *American Education Research Journal, 23*, 248- 262.

Acknowledgments

This material is based upon work supported by the National Science Foundation (NSF-CDI-1027736) to Mitchel Resnick, Yasmin Kafai and Yochai Benkler. The views expressed are those of the authors and do not necessarily represent the views of the National Science Foundation, Utah State University, or the University of Pennsylvania.