Intern. J. Comput.-Support. Collab. Learn (2015) 10:425–447
DOI 10.1007/s11412-015-9217-z

AMOEBA: Designing for collaboration in computer science classrooms through live learning analytics

Matthew Berland[1] · **Don Davis**[2] · **Carmen Petrick Smith**[3]

Received: 12 October 2013 / Accepted: 30 June 2015 / Published online: 1 August 2015

Abstract AMOEBA is a unique tool to support teachers' orchestration of collaboration among novice programmers in a non-traditional programming environment. The AMOEBA tool was designed and utilized to facilitate collaboration in a classroom setting in real time among novice middle school and high school programmers utilizing the IPRO programming environment. AMOEBA's key affordance is supporting teachers' pairing decisions with real time analyses of students' programming progressions. Teachers can track which students are working in similar ways; this is supported by real-time graphical log analyses of student activities within the programming environment. Pairing students with support from AMOEBA led to improvements in students' program complexity and depth. Analyses of the data suggest that the data mining techniques utilized in and the metrics provided by AMOEBA can support instructors in orchestrating cooperation. The primary contributions of this paper are a set of design principles around and a working tool for fostering collaboration in computer science classes.

Keywords Computer science education · Learning analytics · Classroom orchestration · Constructionism

Basic computer skills are not enough to meet the needs of today's increasingly technical society. The ability to harness the computational power of computers for creating, editing, and analyzing information is a necessary skill for many professions, making computational literacy

✉ Matthew Berland
mberland@wisc.edu

Don Davis
dondavis@reglue.org

Carmen Petrick Smith
carmen.smith@uvm.edu

[1] Department of Curriculum and Instruction, University of Wisconsin–Madison, 225 N. Mills Street, Madison, WI 53706-1795, USA

[2] Department of Interdisciplinary Learning & Teaching, University of Texas at San Antonio, San Antonio, TX, USA

[3] Department of Education, University of Vermont, Burlington, VT, USA

an essential component of a 21st century education. However, we have long known that learning computer programming is not easy (Soloway and Spohrer 1989). Traditional instructional approaches involve students working individually manipulating simple content such as numbers or strings (Guzdial 2007). As computer science (CS) educators work to reform curriculum, there has been an emphasis on collaborative (Braught et al. 2011; Preston 2005; Sanders 2002) and non-traditional programming environments (Berland et al. 2013b; Blikstein et al. 2005; Kelleher and Pausch 2007). In this work, we attempt to expand ways in which learning analytics tools can better support teachers as they use constructivist or constructionist curricula. To that end, we developed AMOEBA, a tool to help teachers make real-time decisions about how to best group students to allow for productive collaboration in a non-traditional programming environment. AMOEBA provides multiple unique affordances for facilitating and evaluating collaboration. These affordances build on data mining analyses that provide real time metrics for identifying potentially successful partners and for determining the success of pairings including measures of participation and learning transfer. In this paper, we will outline the design principles that guided the development of AMOEBA, describe a user study, and evaluate the design of AMOEBA both for adherence to design principles and the effectiveness in helping students learn and helping teachers teach. The primary contributions of this paper are a set of design principles around and a working tool for fostering collaboration in computer science classes.

Co-constructionist design principles for CS classroom orchestration (C3P)

There has been an increasing push towards collaboration in CS for many reasons. Much of the motivation arises in attempts to curb the significant underrepresentation of many demographic groups in CS including women and ethnic minorities (Margolis et al. 2008). Collaboration in the classroom has been noted as especially significant for underrepresented students in CS (Li et al. 2013; Werner et al. 2004) as it may contribute to a "sense of belonging and security" (McKinney and Denton 2006, p. 138), the lack of which represents a significant deterrent to underrepresented students in CS (Margolis and Fisher 2003b; Teague 2009). As such, we have distilled existing literature on design for collaboration and computer science education to four core design principles:

C3P 1. Iteratively integrate feedback from working CS classrooms.
C3P 2. Optimize for student and teacher co-construction of mutually useful artifacts.
C3P 3. Maximize meaningful student-student interaction around data-rich artifacts.
C3P 4. Use analytics to leverage students' different skills and proficiencies.

C3P 1: Iteratively integrate feedback from working CS classrooms

One of the most salient ways to address real problems in computer science classrooms is by helping real teachers. Teachers in K-12 CS classrooms have only limited access to resources, training, other teachers in the same discipline, or outside support (Guzdial et al. 1997). That said, a variety of research over many years has shown that responsive teaching can radically alter both class makeup and class outcomes (Darling-Hammond 1997). Supporting CS teachers in the classrooms can be hard, as features specific to CS classrooms can make it

difficult to teach (Ben-Ari 2001): the content is technical; students often look at screens through class; program code takes a lot of time to read, and it can be intricate; content and contexts change frequently; and few material resources explicitly support CS teachers' use of collaborative approaches. Moreover, many of the existing advanced tools for teaching computer science are designed for content experts rather than expert teachers (Koehler and Mishra 2005) – a deficit that is exacerbated in CS classrooms as content experts are much more likely to be hired away by industry before they become expert teachers (Ericson et al. 2007) .

Consequently, to be a useful tool for a real classroom, the tools must lend themselves to good teaching practice, be accessible to expert teachers who are not themselves content experts – i.e., they must be *teacher intuitive* and *teacher compatible*. More importantly, the tools must not get in the way of good teacher practice: they must allow teachers to see students' faces; not require the full attention of either teachers or students; and allow for movement and group work. In adhering to this principle of design in and with real classrooms, a small tradeoff is made. It is easier and often more fruitful to iterate on design that is tested in a real-world setting, even if it "muddies" results more than a carefully controlled lab study might. This deference to naturalistic testing is a core tenet of design and design-based research (Barab and Squire 2004; Edelson 2002).

C3P 2: Optimize for student and teacher co-construction of mutually useful artifacts

Constructionism (Papert and Harel 1991; Papert 1980) is a *framework for action* (diSessa and Cobb 2004) for constructivist design with real learners. Constructionism has proven to be a remarkably helpful model for teaching computer science (Papert 1980). This may be due to a fundamental connection between learning through the programming and design of artifacts (as per constructionism) and teaching in a discipline in which programming and the design of software artifacts are part of the broader curriculum. That said, even many good constructionist projects can insufficiently motivate the "shared meaning" aspects of constructionism (Papert and Harel 1991; Shaffer and Resnick 1999). By emphasizing social elements of programming projects, i.e., creating with and around other people, students may become more engaged and achieve greater mastery of the content (Fosnot 2005).

C3P 3: Maximize meaningful student-student interaction around data-rich artifacts

"Meaningful collaboration" is purposely bidirectional: the collaboration itself should be salient and useful to the participants and the collaborative work should focus on projects personally meaningful to all participants. Researchers have noted various benefits to collaboration – and pair programming in particular – including: improved program quality (Nagappan et al. 2003); increased understanding and transfer of programming concepts (Braught et al. 2008); improved engagement with; and more positive affect towards programming (Martin et al. 2013). For paired programming to support learning, students must be paired effectively. Often this pairing is done by students themselves or by teachers without referring to data. The assessment of group and pair efficacy in CS often only occurs across long time frames, meaning that an ineffective pairing might not be discovered until the end of a project (Srikanth et al. 2004). Research on collaborative learning in other settings has also shown that dysfunctional pairings can hinder collaborative learning (Salomon and Globerson 1989). As such, determining how to create pairs that support collaboration, learning, and engagement is a critical challenge for CS teachers.

Many researchers have indicated that skill level is an important factor in creating successful programming pairs (Cao and Xu 2005; Chaparro et al. 2005; Sanders 2002).[1] However, identifying a student's skill level is not a straightforward task as there has been little agreement on adequate measures of CS skill suitable for such purposes. To date, researchers have sought various methods to identify programming skill similarity (e.g., Cao and Xu 2005; Chaparro et al. 2005; Katira et al. 2004; Radermacher et al. 2012; Watkins and Watkins 2009) with varying results. A common approach to quantifying programming skill, has been to rely on course grades (Cao and Xu 2005; Watkins and Watkins 2009), though grades do not correlate significantly to either CS or programming skill (Byckling and Sajaniemi 2006). Other researchers have used more elaborate measures to identify students' skill. For example, Katira et al. (2004) utilized midterm grades, GPA, and SAT scores, which offer only limited help in identifying programming skill (Byckling and Sajaniemi 2006). Moreover, such overarching, broadly encompassing measures shed little insight on relationships among CS partner pairing, learning to program, or computational literacy more broadly.

One aspect affecting the accessibility of student-student interaction is also referred to as 'visibility' (Lyons et al. 2015). That is, participants (both students and teachers) must be able to discern, with minimal effort, how to begin collaborative engagement. This is closely related to work in science education by Roth et al. (1996), who indicate that students most frequently engage in "meaningful" sensemaking practices (around computers and computation) when students perceive clear value in task completion.

C3P 4: Use analytics to leverage students different skills and proficiencies

Given existing findings on collaboration in computer science contexts (e.g., Chaparro et al. 2005; Katira et al. 2005; Van Toll et al. 2007), educators and researchers may benefit from utilizing a more appropriate operationalization of Vygotsky's (1978) Zone of Proximal Development specifically for learning Computer Science ("CS-ZPD"). As in all domains, the tools and processes in which they engage differentiate the material ways in which students collaborate and teach each other in computer science. Perhaps because many tools exist for computer programmers to collaborate in pairs but few tools exist to help programmers learn together, processes of collaboratively learning computer science (especially in pairs) may not correspond exactly to collaboration from other domains. In particular, much of the focus in collaboration around programming focuses on programming in pairs.

As such, computer science education researchers (e.g., Katira et al. 2004; Salleh et al. 2010) have sought to identify partner characteristics that promote successful pairing and have worked to assess the impact of varying student characteristics on pair work, e.g., the effects of similarities and differences among socio-cultural characteristics, such as, personality traits, as well as measuring the effects of cognitive elements, such as, "programming mental models," programming skill, or proficiency. Though various readiness, aptitude, and personality tests may highlight characteristics potentially relevant to successful pairings (Radermacher et al. 2012; Salleh et al. 2010), there are a myriad of other confounding factors e.g., ethnicity and gender that may impede or benefit collaboration (Katira et al. 2005).

[1] Within the literature of pair programming understanding, competence, aptitude, and skill are frequently used near synonymously. Here the term "skill" will be used for this amalgam and the term "proficiency" will be used for a superset of skill and contextual comprehension.

As with other collaborative learning endeavors, collaborative programming suffers commonly from "sucker" and "free riding" effects, whereby difficulties and incompatibilities arise as students respond to and exploit perceived skill discrepancies; for example relying on more skillful students to solve problems and complete assignments (Kangas 2004). Though students frequently indicate a preference for working with other students of similar skill in pair programming groups (e.g., Cao and Xu 2005; Katira et al. 2004, 2005), a difference in skill may be beneficial (Chaparro et al. 2005; Van Toll et al. 2007).

Katira et al. (2004) equates such findings to observed benefits of utilizing ZPD for partner pairing, which aligns such discussions with a history of educational discussion emphasizing the importance of students' ZPD (Fosnot and Perry 2005). Though Katira et al. (2004) indicate that while students significantly indicate a preference for partners of perceived similar proficiency, they do not indicate such a preference for such similarity of technical skill level as measured by midterm grades. That said, grades are not a reliable indicator of students' programming understanding (Byckling and Sajaniemi 2006). Radermacher et al. (2012), noting this, suggested a more CS-specific post-hoc assessment of ZPD, but we found no prior work that attempted to operationalize anything similar to *live* CS-ZPD.

CS-ZPD: Operationalizing our design principles

Operationalizing the preceding design principles is not straightforward. Vygotsky considered measures of an isolated individual's problem solving skill as inadequate; rather, it was more meaningful to determine what a student could accomplish with a more capable peer (Fosnot and Perry 2005, p. 23). As such, any tests taken in isolation and not measuring specific learning changes in relation to work with others are at best, predictive estimates of students' potential ZPD. If CS-ZPD is to be operationalized further, more nuanced methods, preferably over time and in consideration of pair effects, such as our "surprising program similarity" metric, may be more appropriate for CS and programming should be identified.

Our project uses a metric of "surprising similarity" because it suggests that two students are solving a problem in similar ways and that those similar problem approaches are not themselves common. In Berland et al. (2013a), we describe this in terms of the programming language, C++:

> Program code, even in a limited language like IPRO, can vary enormously. In C++, there are infinite variations of source code that could produce 'hello world'. That said, (almost) every one of those programs will have 'int main (…)' at the beginning – that code is common to almost every C++ program ever written. By using a similarity metric that matches isomorphic code but discounts code provided by teachers, required code, common code, or obvious code (such as 'int main (…)' in C++), we can find students who are using similar logic and approaches.

Matching students based on their similar approaches is situated in our operationalizing of ZPD. In particular, we suggest that:

1. Students perceiving and attempting to solve a problem in similar ways – as per our surprising similarity metric – will be able to understand and support each other more effectively than students who have trouble understanding each others' approach;
2. This will be especially true across students having different levels of success (as described by *program quality*, another metric described below); but

3. This is not desirably measurable in a lab (or randomized control) setting, so data must be authentically contextual; and
4. This tool for pairing will be both authentically useful and meaningful to an actual teacher.

There is a variety of supporting evidence for these suggestions, including students' improved effectiveness in communicating and co-constructing of understanding owing to their ability to perceive the problem similarly (Goos et al. 2002, p. 196).

Fundamentally, however, this work is not theory verification but rather design-based research (Barab and Squire 2004), in that we are evaluating whether and how a specific design (organized around a set of design principles) affected actual classroom environments. *Teacher compatibility* (as per CS-CPD 1) is paramount. As per CS-CPD 3, matching students based on similarity of code is not necessarily the best way of creating groups for every group in every classroom – that is far more contextual than provable and no such thing likely exists. However, it is a very *comprehensible* way of matching students, it has both theoretical and face validity in that it is easily explicable to teachers, and, at worst, prompts teachers to create structured groups where there had been none (which, as we shown above, has generally positive effects).

Note that the metrics are described in technical detail in the *Measures* section below.

Mining collaboration

Large scale, yet nuanced inquiries related to facilitating classroom collaboration, similar to the AMOEBA project described here, are gaining support from data mining and learning analytics researchers (Anaya and Boticario 2011; Talavera and Gaudioso 2004). Owing to greater accessibility to vast amounts of data and data processing power previously untenable, data analysis tools and techniques are being increasingly adopted in learning analytics approaches to better identify and disaggregate potentially significant learning trends within large corpora of data (Baker and Yacef 2009; Romero and Ventura 2010). Grounded in such practices, researchers have pointed out potential benefits to facilitating and analyzing collaboration with data mining techniques (Gaudioso et al. 2009). As unstructured environments may be cumbersome to analyze due to difficulties in parsing natural language (Soller et al. 2005), researchers have sought and developed alternatives to unstructured text in order to simplify natural language processing (Barros and Verdejo 2000). These include providing established prompts and sentence frames to students to articulate their responses, which substantially lightens the difficulty of analysis without significantly detracting from cognitive content (Barros and Verdejo 2000).

Researchers have indicated that such data mining and presentation of concomitant findings in an instructor accessible manner may serve to support teachers in better orchestrating successful student collaboration (Soller et al. 2005). In doing so, it is important to acknowledge that given classroom dynamics, it often becomes important for teachers to utilize a plethora of grouping strategies (Jermann et al. 2002) and that technology supporting these groupings should afford teachers' significant discretion and flexibility in how such technology is utilized (Dimitriadis 2012). Therefore, in order to support teachers while allowing flexibility, tools to support teachers' orchestration of programming collaboration should perhaps initially focus on providing data mined metrics in real time that determine whether learning has occurred, particularly as evident through transfer, and that measure student participation (Dillenbourg et al. 2009).

Novice programming environments

As an effort to make programming more accessible and to increase participation in CS, several alternative programming environments have been developed. These environments, rooted in constructionism, forgo the text-based syntax of traditional programming languages in favor of visual and media rich interfaces. Environments such as NetLogo (Wilensky 1999), Scratch (Resnick et al. 2009), and Alice (Kelleher and Pausch 2007) have successfully supported programming novices' learning and engagement. Building on research into these environments, we developed IPRO, a mobile, visual, and social programming environment for novice programmers (as young as seventh grade) to use on iOS devices such as iPads or iPhones (Berland et al. 2011).[2]

Specifically, the IPRO environment was developed with a focus on collaboration, tinkering, ease of creating simple working programs, and mobility, as evidenced through participation and direct embodiment (Berland et al. 2011). For ease of use, the IPRO environment relies on students' drag and drop programming to control virtual soccer playing robots (for solo or team play). The visual drag and drop programming block structure of IPRO enables novice programmers to complete programming tasks of greater complexity than might be otherwise feasible. This structure eliminates the possibility of syntax errors.

Perhaps most substantively, IPRO's design is heavily informed by constructionism (e.g., Papert and Harel 1991) and embodied cognition (e.g., Abrahamson 2009). With regards to its constructionist underpinnings, IPRO is designed to build upon students' inclination to explore programming concepts in order to create artifacts (here, virtual soccer playing robots) that are shared with one another in a public space, which results in greater engagement and concomitant learning (Papert 1980). The mobile aspect of the IPRO environment is, in part, a divergence from the "driver/navigator" model of CS pairing, whereby one student enters commands and the other oversees the work (Van Toll et al. 2007). Rather, it is intended that each student having her own device should bolster active participation and reduce many of the problems frequently occurring with collaboration, such as "sucker" or "free-rider" effects. Together the structure of IPRO and the logging of each state of students' programs as they are being edited supports real time analyses of students' programming and programming trajectories more than currently achieved elsewhere (Berland et al. 2013a). Further discussion of the affordances and design considerations of IPRO are provided elsewhere (e.g., Berland et al. 2011).

AMOEBA

The AMOEBA tool provides real time analyses of students' programming behaviors within the IPRO visual programming environment in order to support teachers in orchestrating classroom collaboration. It uses the IPRO data logged in real time on a network server to give teachers information about students' programming progression (uniqueness and proficiency) to inform the creation of effective programming pairs.

In order to meaningfully frame understandings of AMOEBA and IPRO, but not exceed the scope of this paper, it is important to note the key motivations and theoretical basis of this and much other collaborative work. As with much other pair programming research (e.g., Katira et al. 2004), work with AMOEBA and IPRO builds substantively on constructivist notions of

[2] IPRO is available free of charge on the iTunes store.

learning (Fosnot 2005). Namely, IPRO and the AMOEBA tool are informed by understandings that learners construct knowledge and skills through piece-wise building on previous learning and experiences. Moreover, as neither Piaget nor Vygotsky considered these processes purely cognitive or purely social (Tudge 1992), nor do the authors of this paper. Similar to other collaboration researchers, the authors of this paper build on notions that students may learn more in conjunction with others (Dillenbourg et al. 1995; Katira et al. 2004; Soller et al. 2005). Additionally, these constructivist understandings are further informed by research indicating the learning affordances of collaborative work that allows for physicality, such as gestures in supporting learning (Singer et al. 2008) and related research highlighting the affordances of "embodied cognition" in facilitating learning within novice programming environments (Petrick et al. 2011). However, moving beyond such understandings, the work with AMOEBA and IPRO builds upon more specific understandings of constructionism emphasizing the importance of shared artifacts (e.g., programs) in a public space, such as a robot soccer tournament (Berland et al. 2011).

AMOEBA's unique affordances in supporting collaboration

AMOEBA's novelty arises extensively from its ability to provide real time analyses of students' programming behaviors in order to support successful collaborations. Real time analyses are only as beneficial as the measures they present and to the extent that the measures are presented meaningfully. These relate substantively to how (and why) AMOEBA is used in supporting teachers' orchestration of classroom collaboration. AMOEBA is unique in supporting collaboration by providing real time metrics to be acted upon as the teacher sees fit to orchestrate classroom-programming collaboration. In this study, we focus on the impact of utilizing a unique measure that roughly equates to a 'operationalized ZPD for novice programmers' to orchestrate collaboration.

Though researchers have indicated the benefits of utilizing real time analyses to support learner collaboration (Dillenbourg et al. 2009), there are few tools which do so (cf. Bachour et al. 2008). Moreover, no tools were identified that provide such real time collaboration *orchestration* support in novice CS classrooms. Although tools were identified that facilitated collaboration in novice CS classes (Flieger and Palmer 2010), allowed analysis of learners' behavior in open programming environments (Blikstein 2011), and allowed for analyzing asynchronous collaboration among more advanced programmers and programming students (e.g., Anaya and Boticario 2009), no such tools were identified for novice visual programming environments. Making these analyses in real time is critical in a novice-programming environment as dysfunctional collaborations may impact students' impressions of programming. The first impressions engendered in such environments are significant as it is within such environments that many students may decide if they can or should (continue to) study computer science (Margolis and Fisher 2003a).

Operationalizing CS-ZPD in AMOEBA

AMOEBA utilizes a "surprising similarity" metric to identify uniqueness in students' programs. This is based on Dunning's (1993) metrics of surprise and coincidence, which are measures of inverse-log-likelihood. This is similar to *tf-idf* (Salton 1989), which is the product of similar elements multiplied by the inverse frequency of that element within the corpus as a whole. We describe more technical details in the Measures section below.

This metric is used in real-time evaluations of students' programs, which, consequently, provides actual rather than predictive estimates of students' current programming status. As such, the measure provides an indicator of similar or unique approaches to programming within the environment. On the basis of these measures, AMOEBA provides the initial recommendation for student pairings. This highlights that students are within one another's ZPD, though it does not indicate the 'higher' or 'lower' of the two. As students work on their programs, AMOEBA updates its pairing recommendations in real time; teachers have the option to maintain pairings that are functioning successfully or repair students based on AMOEBA's updated pairings. Currently, AMOEBA does not evaluate skills transfer or participation in real-time. The study was primarily concerned with the feasibility and impact of utilizing real-time CS-ZPD scores.

Research questions

Consequently, our research sought to answer the following questions:

1. How does the utilization of real-time CS-ZPD scores as the basis for student pairing in a novice CS class impact students' programming performance (e.g., program rarity, quality, depth, and specificity)?
2. How do such effects (if any) relate to level of CS-ZPD similarity?
3. How does our work meet the design principles identified above?

Methods

Overview

In this study, we collected data from students, across three sites and eight classes, with little or no programming experience. Students were placed in pairs on the basis of predictive CS-ZPD as indicated by AMOEBA. During the course of single programming sessions, most commonly approximately 45 min in length, students were asked to collaborate based on the predictive CS-ZPD score. As programming data was generated and analyzed, some students were re-paired with others with a greater shared CS-ZPD score. Students' program data were analyzed to explore how pairing with AMOEBA impacted: the complexity of students' code; program novelty; and program quality.

Participants

The only explicit limitation to participant selection was the requirement that students be in the seventh grade or higher as a pilot study indicated that younger students struggle disproportionately with the IPRO interface. Sites were selected opportunistically through researchers' direct or indirect relationships with the coordinating teachers and administrators. Then, participants were selected on an opportunistic basis from volunteers across the multiple sites; three willing technology teachers were found who agreed to participate in the study. Their students who agreed to participate were allowed to do so.

For this study, junior high and high school students (n=95) engaged in IPRO programming activities. In the course of which, 70 of these students were recorded working in pairs as facilitated by the AMOEBA interface. Students from eight different classes across the three different sites participated: one class as part of an extra-curricular lunch time activity for 7th and 8th grade students at a private school for gifted children in Wisconsin (n=14 total), six classes as an assignment alternative for 7th and 8th grade students in a middle school technology applications class in central Texas (n=60 total), and as part of a voluntary enrichment activity in one first semester computer science class at a central Texas high school with students in the 10th grade or higher (n=21 total).

The student population of the private school for gifted students in Wisconsin is more ethnically diverse than the majority of surrounding public schools. At the central Texas middle school, student demographic populations are labeled as approximately 13.1 % African-American, 2.9 % Asian, 32.1 % Hispanic, and 51.4 % white, with approximately 26.2 %% of the school receiving free or reduced lunch. At the central Texas middle school, 'tech apps' is a mandatory course for all students; consequently, participant demographics can be reasonably expected to reflect this ratio. The central Texas high school with participants enrolled in the Advanced Placement computer science course, AP CS1, has a greater than 70 % Hispanic student population, approximately 20 % white student population, approximately 6 % African-American student population, and approximately 1 % Asian student population, whereby more than 60 % of students receive free or reduced lunch. However, only an estimated third of CS students were Hispanic with instead a preponderance of white males.

Setting

At the Wisconsin site, volunteer 7th and 8th grade technology students joined in the activity during their lunch time (many while eating lunch) situated in a semi-circle in the front half of a technology application classroom with the IPRO tarp (a real world representation of the IPRO game space) to the side. At the central Texas middle school, 7th and 8th grade volunteers gathered in a circle in one half of their technology application classroom around the IPRO tarp during class time (six classes participated) while other students, separated by rows of computers, engaged in classwork. At the central Texas high school, all students in the AP CS1 class participated and moved about in a more 'typical' classroom with desks that were pushed back to allow greater movement with the IPRO tarp in the center.

In all of the settings, students used iOS devices to work with the IPRO visual, 'drag and drop', programming environment. At the Wisconsin site, students used their own devices as well as iPods provided by the researchers. At the central Texas sites, students either used iOS devices provided by researchers, provided by the school, or their own devices. At the central Texas high school, all students in the CS class volunteered to participate. At the central Texas middle school, problems with Internet connectivity limited the number of students who could meaningfully participate; only ten students (at best) could connect to the internet during each class in addition to those students with iOS capable phones. Any participant with their own iOS phone (and parental assent) was allowed to participate; others were selected on a first come basis.

Data collection

Data was collected at all three sites by the same author and supported at the Wisconsin site by a graduate student. Various metrics were used for evaluation (see below). Researchers examined

program rarity, quality, depth, and specificity [described below]. Two sources of data were utilized in this study: programming data and pairing data. The programming data consisted of automatically logged program edits within the IPRO environment – in IPRO every program edit is automatically saved to the IPRO server. The pairing data was provided and tracked by AMOEBA, which provides a graphical representation of students' program similarity based on real time analyses of IPRO data and allows the facilitator to 'tap' (or click) to record student pairings. Additionally, a backup paper log was kept of pairings to insure accuracy.

Materials

Within the IPRO programming environment, students use visual programming blocks to construct programs (see Figs. 1 and 2). These blocks consist of logical elements such as 'AND' and 'OR' statements, that are combined with actions elements (e.g., go forward-left) to make programs. For example, a student might write simple programs such as: "IF ball is forward-left, THEN go forward-left, ELSE turn left" and increasingly complex programs such as "IF ball AND goal are forward-left, THEN go forward-left, ELSE IF ball is forward-left AND goal is not forward-left, THEN go backwards, ELSE IF…" and so forth. Further discussion of the affordances and specifics of IPRO are available elsewhere (Berland et al. 2010). IPRO automatically logs every program edit each student makes to the IPRO server.

The AMOEBA tool utilizes real time analyses of students' programming in order to inform teachers pairing decisions. AMOEBA provides a graphical representation of students' program similarity based on real time analyses of IPRO data (Fig. 3). To do this, AMOEBA utilizes algorithms to identify novel code sequences particular to students' programming understandings. Once identified, these elements of students' code were utilized to provide a predictive CS-ZPD metric by which to pair students. The graphical representation draws a line connecting students within each other's CS-ZPD, indicating a possible pairing. If the teacher pairs the students, he or she taps or clicks on the two students in the graphical representation to record the pairing.

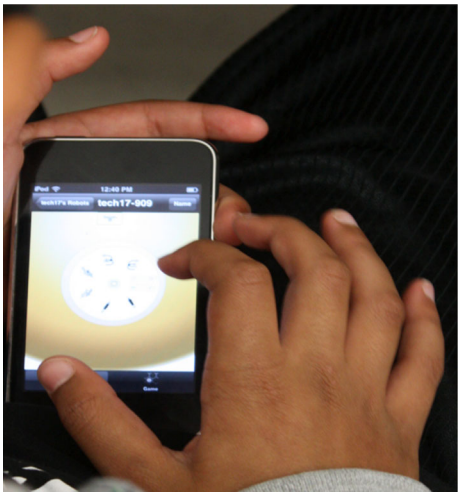

Fig. 1 A student programs IPRO

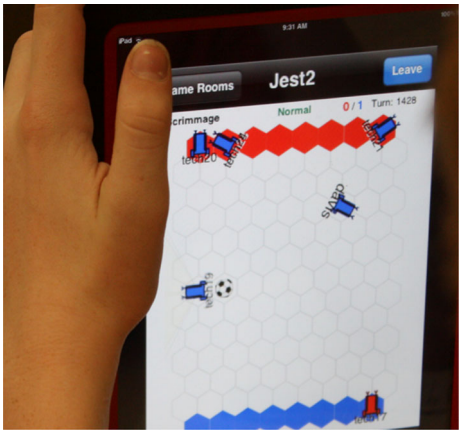

Fig. 2 A student watches her IPRO program run

Procedures

The same procedures were used with all three groups of students including three different activity phases. The first activity involved a teacher-led introduction into the IPRO programming environment in order to acquaint students with the IPRO environment and the intended goals of the exercise (i.e., program robots to score goals and collaborate with partners to do this better). The second activity provided students a short time to program by themselves and gain greater familiarity with the IPRO programming environment. This also provided the data necessary to make initial predictive CS-ZPD analyses to be used to pair students. The third activity began after the teacher made the initial pairings and continued until the end of class. When the teacher made a pairing, he instructed the students that their programs were similar and asked them to work together. The students then moved so that they were sitting near each other. They continued to work on their individual programs, but they were asked to help each other with the problems they were facing. The students showed their partners their programs and asked for help improving them. During this phase, the instructor used the AMOEBA tool to monitor and pair students. The instructor monitored AMOEBA for increased shared novelty indicated by red (minimal meaningful connection), yellow, or green (highest level of shared novelty) lines.

During this time, the facilitator paired and reassigned pairs in response to students' current shared novelty; green lines were given preference over yellow and yellow over red. Students were assigned new pairs as connection strength between partners faded or stronger connections with others became apparent. For example, a student previously paired opportunistically would be reassigned to a student with whom she shared a 'red' connection and then reassigned to a 'green' connection if one appeared. Similarly, students that may have initially been paired on a 'yellow' connection might be re-paired with 'red' connections upon the disappearance of their connection. For those students who were re-paired on the basis of a higher color status, their pair status remained 'paired'. Students who had been paired and were no longer with partners, had pair statuses of 'between pairing' or 'after pairing' depending on whether they were later assigned another partner. The instructor was mindful that rapid repairing might have a negative impact on student learning; thus, the instructor looked for especially strong similarity metrics before re-pairing a student.

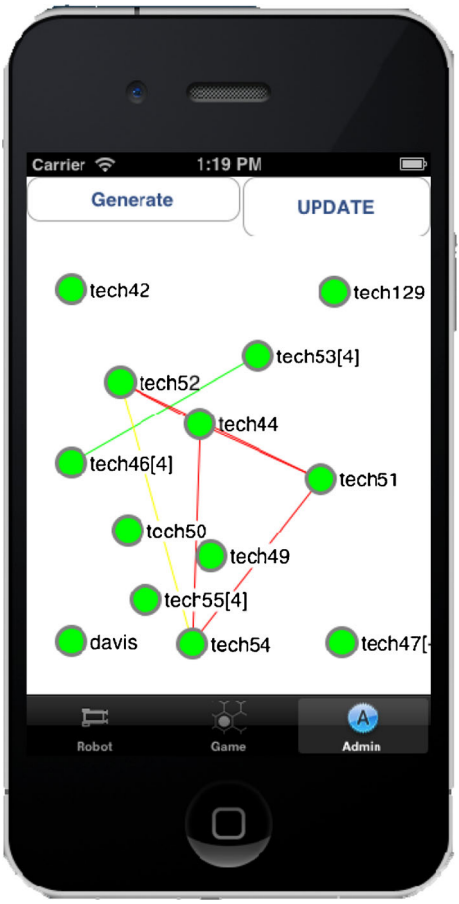

Fig. 3 The AMOEBA interface shows students as 'nodes' and similarities as the 'edges' between them

Measures

We used the SPSS advanced statistics software package with a repeated measures design to look at the effect of *pair status* [pre-paired, paired, post-paired] on four outcome variables [*rarity*, *depth*, *quality*, and *specificity*] (described below) by *student* (n=95) across 3 sites. We are using the same depth, rarity, and quality metrics described in detail by Berland, Martin et al. (2013), but we will describe them in this work in a more abbreviated way.

Rarity is the mean "unlikeliness", on a scale of 0 to 1, of a given program's sub-trees of its parse-tree compared against all other code ever written in a single class. In English, the rarity of "a hyacinth substantiates programmatically" would be near 1 because the sentence and all of its individual nouns and verbs are rare, while the rarity of "I ate food" would be near 0, as all verbs and nouns are common. The most common programs in a class score 0. Programs that share no sub-trees with any other programs written in that class score a rarity of 1.

Program quality The first measure of *quality* is a measure unique to the IPRO environment. Namely, students' robots are run in simulated environments to determine the quality of their robots in scoring goals (goals for) and in preventing goals by opposing robots (goals against).

An average number of goals for and goals against over 250 simulated games of 200 turns each in order to provide a rough estimate of program quality (see Berland et al. 2013a). It is robust, but the quality of programs written by novices in less than 90 min tends to skew negative, because, although it is easy to write a functioning program in such a short time, it is difficult, by design, to write an ideal IPRO program.

Depth, as used to discuss IPRO and other programs, is the number of levels of *parse-tree* (as per Purtilo and Callahan 1989) that the compiler is required to traverse (see Fig. 4). All IPRO programs converge to a single parse-tree, whereby the 'levels' (or 'height') arise from the use of varying 'IF' branches. Each defined condition (e.g., if some condition, then do some action) adds an additional layer. Consequently, complex logic that accounts for more conditions and possibilities will generate a deeper parse-tree. As a result, simple programs have shorter parse trees, and more complex programs have greater depth. Though complexity may not necessarily indicate competence, writing a program in IPRO that generates a deep parse-tree (greater than, say, 5) is a difficult task, and is extremely unlikely to be achieved through chance or randomness, as the parse-tree will prune redundancies.

Specificity is a simple tally of a program's length. Whereby the *depth* metric counts levels of branching conditions (IF statements), the specificity metric simply denotes programmatic descriptiveness. Each detail added to an IPRO program, regardless of type (IF statements as well as 'AND' and 'OR' logical descriptors), creates an added nest of specificity. A plain text example might be: "If the ball is to the right, then turn right," which would have the lowest possible specificity score of 1. By contrast, "if the ball is to the right AND the opponent is to the left, then turn right" would have a specificity score of 2 owing to the addition of an extra descriptor. For pragmatic reasons, only those programs with specificity >1 were evaluated. The rationale being that it was considered important to compare the progress of those students actually doing things and to not potentially skew the data with empty clicks generated by students experiencing Internet connectivity or other issues. A *specificity* score greater than 1 is the first length of a program that could meaningfully indicate anything. Specifically, length refers to numbers of parentheses in the log data for each program edit - one set of parentheses or fewer is merely a click on a screen without attempting to program anything.

Similarity AMOEBA creates a *link* (a visible line) between students based on the similarity of their code, and, in particular, all the sub-trees of the parse tree of their code. That is, each student's code is parsed out into a tree, and then all sub-trees of that parse tree are enumerated. The similarity metric is the maximally surprising sub-tree match between two students. Two students whose code has no isomorphic sub-trees (i.e., no sub-trees with the same semantics) have a similarity of zero, and this would be represented in the AMOEBA interface by a red line. Two students who share a whole program that no other student has ever written would have a similarity of one, which result in a green line connecting the two students. The

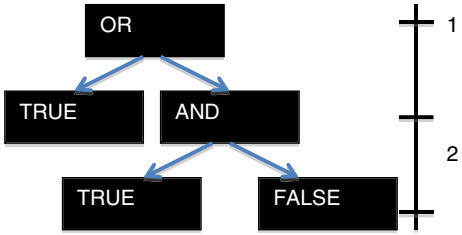

Fig. 4 A logical parse-tree with a depth of 3

similarity metric itself is based on *surprise* (Dunning 1993). Dunning (2008) describes the core elements of surprise:

> The method at the heart of [surprise] is to use a score to analyze counts of events, particularly counts of when events occur together. The counts that you usually have in these situations are the number of times two events have occurred together, the number of times that they have occurred with or without each other and the number of times anything has occurred. Informally, it can be thought of as the unlikeliness that two similar pieces of code happened by coincidence.

The log-likelihood ratio (or *LLR*) can be implemented in only 2 lines of code, provided in Dunning (2008).

Results

In Table 1 (below), we compared the depth, rarity, quality, and specificity of students' programs based on whether the program was created while the student was working without a partner (unpaired), currently working with a partner (paired), or had previously worked with a partner but were currently working alone (after pairing). The changes in all four metrics over time are each individually significantly different by pairing condition, when controlling for covariance within group and individual (as per Fig. 5). That said, the effect of pair status on each individual measure was small, as one can see in Table 1. Table 2 shows aggregate statistics for all measures, for reference.

As shown in Figs. 5 and 6 and Table 1, *rarity*, *quality*, *depth*, and *specificity* increase when students are paired or after pairing. Rarity increases as students are paired and continues to increase after students are no longer paired. Quality increases after students have been paired, and depth and specificity increase when students are paired and maintain a similar level after pairing.

The site effect was quite strong, but recent work in analytics comparing different contexts (e.g., Pardos et al. 2013; Sao Pedro et al. 2013) suggests that time series data can be very sensitive to context. Still, when we account for the covariance within site and student, we see increases in all three metrics.

As previously mentioned, there were significant differences across sites with the most positive gains noted once students had been paired. Additionally, amongst paired students, there were significant effects regardless of whether the AMOEBA connection strength between the two students was red, yellow, or green. Differences regarding quality were similar to data gathered in our initial AMOEBA exploration (Berland et al. 2013b); namely, quality (i.e.,

Table 1 Repeated measures fixed effects by site and pair status [*unpaired*, *paired*, *after pairing*]

	Rarity			Quality			Depth			Specificity		
	n	F	p	n	F	p	n	F	p	n	F	p
site	95	6.04	0.00	95	1.58	0.15	95	11.49	0.00	95	10.27	0.00
pair status	95	3.32	0.04	95	2.44	0.09	95	13.16	0.00	95	9.54	0.00

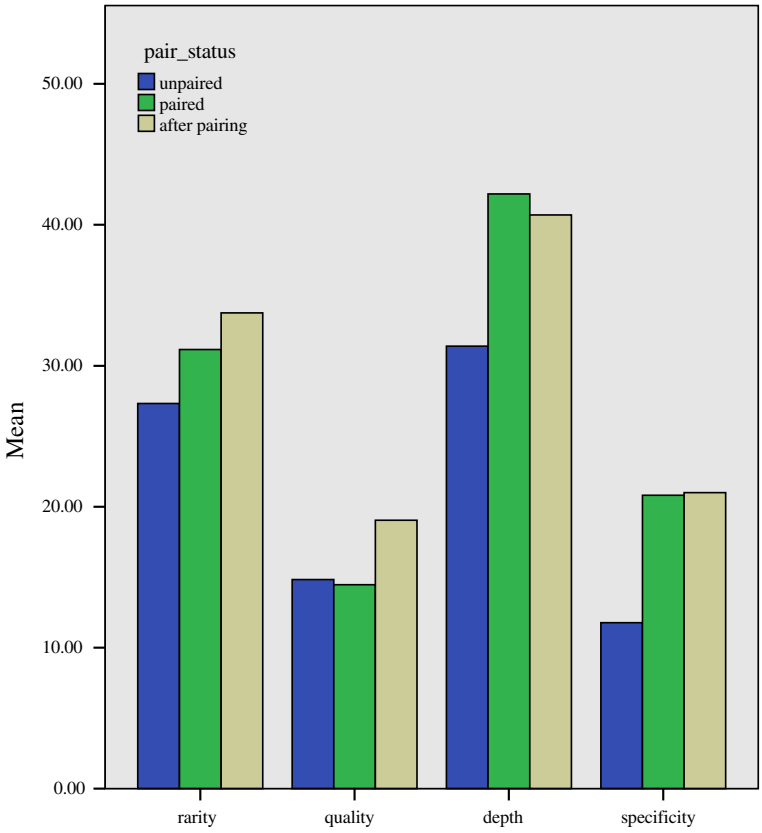

Fig. 5 Rarity, quality, depth, and specificity across all students (n=95), grouped by pair status

'goals for' less 'goals against') seemed to decline by quality of pairing metric - as programs become deeper and more interconnected both goals scored by the bot and goals scored against the bot grow quite substantively, often resulting in a lower goals for ratio than with less complex programs. Here, the quality differences were significant, but not in a visible pattern.

Discussion

While other CS-ZPD metrics may be better than our "surprising similarity" in helping students learn, this study serves as a strong proof-of-concept that pairing students based on CS-ZPD can

Table 2 Aggregate statistics for each measure across all students (n=95)

Rarity		Quality		Depth		Specificity	
Mean	sd	Mean	sd	Mean	sd	Mean	sd
29.59	11.62	15.25	7.23	36.63	16.34	16.34	15.10

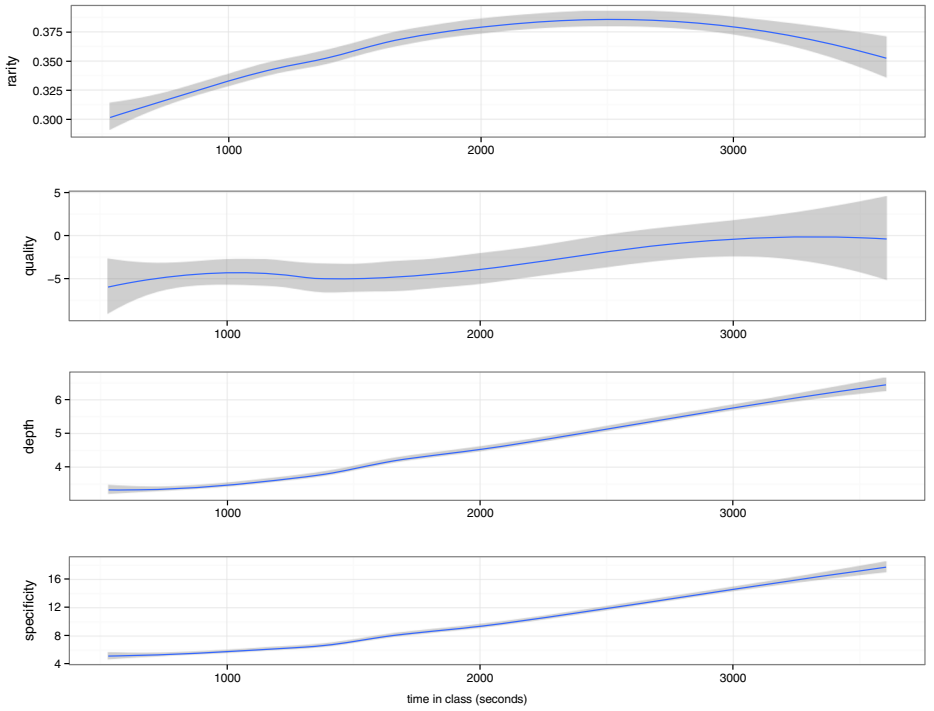

Fig. 6 Graphs of aggregated statistics for each measure over time (loess smoothed)

be helpful. Indeed, we designed this project, in part, to test the hypothesis that giving a teacher or facilitator real-time support about novice students learning to program could help students work together towards more complex work. Across eight separate classes in a variety of settings, our hypothesis seems to be supported by our data. Furthermore, it appears that the design principles that we identified in the literature were helpful in meeting our goals.

Evaluating C3P 1: Iteratively integrate feedback from working CS classrooms

Anecdotally, teachers enjoyed working with the tool, and that is borne out by two minor features of the data: they continued using the tool throughout the class and the students improved throughout. As per our C3P 1, it successfully enabled teaching to do real-time collaborative matching, as they needed: it was designed for them. The specificity of the design challenge will, we hope, make the design more generally useful (as per Barab and Squire 2004). That the design was successful for the context is particularly well supported by a minor feature of the data: they are sporadic. Although the sporadic nature of the data might be considered a weakness in a traditional lab- or user-study, it serves as evidence of quite the opposite here: the tool was a resource – not a driver – of the classroom. This is important, because teachers are more likely to consult valuable resources than replace all class practice with new tools (Mishra and Koehler 2006). In addition, an unobtrusive resource can provide the teacher with a "cognitive handle" – that is, even the best teachers can sometimes neglect or forget to pair students, and this can serve as a reminder. As per our result above, teachers in this study paired their students far more on average than a typical CS class, most of which involve

no pairing (as per Carbone and Kaasbøll 1998). Perhaps most beneficial, students were paired spontaneously on the basis of their approximated CS-ZPD and then demonstrated the improvement suggested by such an approach geared towards leveraging ZPD. Though only a quick approximation, this operationalized CS-ZPD assessment and the surrounding design process highlight possibilities for working with real classroom constraints and opportunities and the real time data available in such environments. That said, we did not exploit all possible avenues for or information in the data we were collecting, because it was made clear through pilot work that teachers favored more targeted and more specific implications from the data analysis. It makes sense: busy experts given a useful tool will enable more immediately productivity than something that requires either training or a shift in strategy.

Evaluating C3P 2: Optimize for student and teacher co-construction of mutually useful artifacts

Students, after having been paired on the recommendations provided by AMOEBA, evidenced more proficient program development – creating more, better programs – than they otherwise might have done. This is not a strongly predictive, prescriptive, or a causal result; this is a relatively small, real-word, design-based research study, and the population is not broadly representative. That said, the results suggest that students' programming improved as they worked together and after they worked together; this suggests that C3P 2 worked in practice for our population. They kept playing, they kept creating, and the classes worked as teams voluntarily throughout. As teachers were more able to coordinate collaboration and see students successfully collaborate with new partners, teacher reported that they could see their role in the construction of the artifact – they felt "useful" to the classroom.

Evaluating C3P 3: Maximize meaningful student-student interaction around data-rich artifacts

To design for meaningful collaboration, we limited the space of possible programs. In the practice of programming in general, there is no implication that any two people (even those working on the same type of project) will have much to code in common. In this study, we reinforced the utility of collaboration by having students work on a variety of different possibilities (goalie, striker, etc.) in the same space with the same code primitives using the same interface. In this way, it became possible to evaluate similarity of code much more easily – the "data-rich" nature of the artifacts enabled us to create collaboration analytics. We could not test an alternative scenario, obviously, because as the projects and environments become variables, many other variables became simultaneously unfixed and confounded. That said, students worked together consistently, and they worked together far more than they do on average in introductory CS classes. However, this is confounded by the use of the IPRO learning environment, which has previously been shown to increase overall student-student collaboration (Berland et al. 2013a; Martin et al. 2013). In those studies, students tend to interact around the "mutually visible" aspects of the artifacts; that is, students collaboratively make sense of visible program elements – they share code, they describe code primitives to each other, and they act out programs.

Evaluating C3P 4: Use analytics to leverage students' different skills and proficiencies

Core to the design of AMOEBA was the idea that students will find their skill levels, but in the pilot, it was very difficult for the teacher to determine that skill level. By adding a rough "evaluative" element to AMOEBA through our analytics – the "approximate quality" metric shown on screen – the teacher could purposefully match students with other students in different skills and skill levels. In the future, similar projects seeking to provide real time valuations of program quality (or complexity, thoroughness, etc.) may also supplement their real-time analytics with "hand-coded" high quality identifiers (e.g., code snippets to match against current student code).

Understanding live collaboration in CS classrooms

Many existing models of computer science education treat code sharing as cheating - it is usually assumed to be detrimental (cf. Katira et al. 2004). Our data suggest the opposite effect: when students shared code, their own individual code got better after that sharing (again as per Fig. 5 and Table 1). To tar CS education with the proposition that all sharing is cheating would obscure much of the excellent work in which secondary school CS and engineering teachers in the US are engaging their students. Further, that means ignoring substantive investigations into pair programming and other collaborative programming practices (e.g., Braught et al. 2008; Guzdial et al. 1997; Repenning et al. 2011; Teague and Roe 2009).

Most (though not all) existing models of assessment in K-12 computer science education rely on relatively few intermediate products evaluated by a facilitator (e.g., Carter 2014). In our assessment, we taught students with a messy creative, open-ended task, and enabled the teacher to assess progress as it happened in class in a repeatable, measurable way. Very few (if any) existing real-world K12 CS classrooms use live dashboards of student progress on code, and we provide one provably workable model. We could find no evidence of any similar, extant, deployable software in our literature review. While not all CS educators will have access to a tool such as AMOEBA, they may still alter their assessment strategies to monitor student work more closely at it is in progress.

The unique CS-ZPD that we sought to evaluate showed some promise. Though the 'quality' metric, as we defined it, did not show a significant positive trend for all of the students, this is perhaps more indicative of a limitation of time constraints and the cost of experimentation to students (Berland, Martin et al. 2013). The other metrics of depth and specificity showed positive differences across the strength of pairing conditions (i.e., typically green>yellow>red). This trend was clearest and most visibly significant for *specificity* in all cases. Similarly, a trend of greater *depth* was evident across pairing conditions, whereby green showed greater depth than yellow and yellow greater than red. Though the difference between yellow and red was not statistically significant given the alpha we selected, the difference may still be potentially informative for researchers (Gigerenzer 2004). In other words, AMOEBA's similarity metric shows potential for identifying students within each other's CS-ZPD. These trends highlight the potential of this approach for pairing students. In the future, it may prove worthwhile to explore the impact of such pairing strategies over longer time and gather more data in order to better support more nuanced examinations of the impacts of such strategies, including the after pairing effects in relation to the operationalized CS-ZPD presented here.

Implications for future work

That students made progress after being paired and subsequently unpaired suggests that there is some preparation for future learning ('PFL', in the sense of Schwartz and Martin 2004) in the act of working on code with peers. That our findings appear robust across multiple similar contexts may be our most provocative finding. It suggests that students are learning something relatively intangible as they write code with friends – something that becomes clearer when they start working alone again (cf. Katira et al. 2004). This corresponds to other understandings of the significance of uninterrupted time for learning to program code and complete other similarly complex tasks (Gillie and Broadbent 1989; Perlow 1999; Speier et al. 2003). While researchers (e.g., Speier et al. 2003) indicate that uninterrupted time is important for cognitively demanding work (such as working out complex code), others (e.g., Katira et al. 2004) show that people learn more complex content together. That our data show this clearly – if weakly – in only very short sessions lending strength to models of CS classrooms such as those of Guzdial and Forte (Guzdial and Forte 2005). It is the consistency of our work with both prior empirical and theoretical work that suggests that this is meaningful progress.

Practically, the relationship of our design principles to our user study findings may have important implications for educators and researchers alike. This work suggests new avenues for designing how teachers assess CS students' collaborative work. Rather than weighing heavily on final products, teachers could assess student work at multiple time points within work on a project, potentially adjusting partners based on the state of the programs. Giving students time to work together, even if they finish their projects individually, appears to have a valuable impact on learning.

Acknowledgments Thanks to the Complex Play Lab for helping refine this work. This work was supported by National Science Foundation Grant No. 1331655. The opinions expressed in this paper are those of the authors and do not necessarily represent those of the NSF.

References

Abrahamson, D. (2009). Embodied design: Constructing means for constructing meaning. *Educational Studies in Mathematics, 70*(1), 27–47.

Anaya, A. R., & Boticario, J. G. (2009). A data mining approach to reveal representative collaboration indicators in open collaboration frameworks. In *Proceedings of the 2nd International Conference on Educational Data Mining (EDM09)* (pp. 210–219).

Anaya, A. R., & Boticario, J. G. (2011). Content-free collaborative learning modeling using data mining. *User Modeling and User-Adapted Interaction, 21*(1–2), 181–216.

Bachour, K., Kaplan, F., & Dillenbourg, P. (2008). Reflect: An interactive table for regulating face-to-face collaborative learning. In P. Dillenbourg & M. Specht (Eds.), *Times of convergence. Technologies across learning contexts* (pp. 39–48). Berlin: Springer.

Baker, R., & Yacef, K. (2009). The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining, 1*(1), 3–17.

Barab, S., & Squire, K. (2004). Design-based research: Putting a stake in the ground. *Journal of the Learning Sciences, 13*(1), 1–14.

Barros, B., & Verdejo, M. F. (2000). Analysing student interaction processes in order to improve collaboration. The DEGREE approach. *International Journal of Artificial Intelligence in Education, 11*(3), 221–241.

Ben-Ari, M. (2001). Constructivism in computer science education. *Journal of Computers in Mathematics and Science Teaching, 20*(1), 45–73.

Berland, M., Martin, T., & Benton, T. (2010). Programming standing up: Embodied computing with constructionist robotics. In *Proceedings of Constructionism 2010*. Paris.
Berland, M., Martin, T., Benton, T., & Petrick, C. (2011). Programming on the move: Design lessons from IPRO. In *Proceedings of the ACM SIGCHI 2011* (pp. 2149–2154). Vancouver.
Berland, M., Martin, T., Benton, T., Smith, C. P., & Davis, D. (2013a). Using learning analytics to understand the learning pathways of novice programmers. *Journal of the Learning Sciences, 22*(4), 564–599.
Berland, M., Smith, C. P., & Davis, D. (2013). Visualizing live collaboration in the classroom with AMOEBA. In *Proceedings of the International Conference on Computer-Supported Collaborative Learning.*
Blikstein, P. (2011). Using learning analytics to assess students' behavior in open-ended programming tasks. *Proceedings of the Learning Analytics and Knowledge Conference (LAK11)*.
Blikstein, P., Abrahamson, D., & Wilensky, U. (2005). Netlogo: Where we are, where we're going. In *Proceedings of Annual Meeting of Interaction Design & Children*.
Braught, G., Eby, L. M., & Wahls, T. (2008). The effects of pair-programming on individual programming skill. *SIGCSE Bulletin, 40*(1), 200–204.
Braught, G., Wahls, T., & Eby, L. M. (2011). The case for pair programming in the computer science classroom. *Transaction in Computing Education, 11*(1), 1–21.
Byckling, P., & Sajaniemi, J. (2006). A role-based analysis model for the evaluation of novices' programming knowledge development. In *Proceedings of the second international workshop on Computing education research* (pp. 85–96). Canterbury: ACM.
Cao, L., & Xu, P. (2005). Activity patterns of pair programming. *Proceedings of the 38th Annual Hawaii International Conference on System Sciences, 2005 (HICSS'05)*, 1–10.
Carbone, A., & Kaasbøll, J. J. (1998). A survey of methods used to evaluate computer science teaching. In *ACM SIGCSE Bulletin* (Vol. 30, pp. 41–45). ACM.
Carter, D. P. (2014). *AP Computer Science: Teacher's guide*. Lancaster: CollegeBoard. Retrieved from http://apcentral.collegeboard.com/apc/members/repository/ap07_compsci_teachersguide.pdf
Chaparro, E. A., Yuksel, A., Romero, P., & Bryant, S. (2005). Factors affecting the perceived effectiveness of pair programming in higher education. In *Proc. PPIG* (pp. 5–18).
Darling-Hammond, L. (1997). The quality of teaching matters most. *Journal of Staff Development, 18*(1), 38–41.
Dillenbourg, P., Baker, M. J., Blaye, A., & O'Malley, C. (1995). The evolution of research on collaborative learning. In E. Spada & P. Reiman (Eds.), *Learning in humans and machine: Towards an interdisciplinary learning science* (pp. 189–211). Oxford: Elsevier.
Dillenbourg, P., Järvelä, S., & Fischer, F. (2009). The evolution of research on computer supported collaborative learning. In N. Balacheff, S. Ludvigsen, T. Jong, A. Lazonder, & S. Barnes (Eds.), *Technology-enhanced learning* (pp. 3–19). Netherlands: Springer.
Dimitriadis, Y. A. (2012). Supporting teachers in orchestrating CSCL classrooms. In A. Jimoyiannis (Ed.), *Research on e-Learning and ICT in Education* (pp. 71–82). New York: Springer.
diSessa, A. A., & Cobb, P. (2004). Ontological innovation and the role of theory in design experiments. *Journal of the Learning Sciences, 13*(1), 77–103.
Dunning, T. (1993). Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics, 19*(1), 61–74.
Dunning, T. (2008). Surprise and Coincidence - musings from the long tail. Retrieved October 5, 2013, from http://archive.is/KH84
Edelson, D. C. (2002). Design research: What we learn when we engage in design. *Journal of the Learning Sciences, 11*(1), 105–121.
Ericson, B., Guzdial, M., & Biggers, M. (2007). Improving secondary CS education: Progress and problems. In *ACM SIGCSE Bulletin* (Vol. 39, pp. 298–301). New York: ACM.
Flieger, J., & Palmer, J. D. (2010). Supporting pair programming with JavaGrinder. *Journal of Computing Sciences in Colleges, 26*(2), 63–70.
Fosnot, C. T. (Ed.). (2005). *Constructivism: Theory, perspectives, and practice*. New York: Teachers College Press.
Fosnot, C. T., & Perry, R. S. (2005). Constructivism: A psychological theory of learning. In C. T. Fosnot (Ed.), *Constructivism: Theory, perspectives, and practice* (pp. 8–38). New York: Teachers College Press.
Gaudioso, E., Montero, M., Talavera, L., & Hernandez-del-Olmo, F. (2009). Supporting teachers in collaborative student modeling: A framework and an implementation. *Expert Systems with Applications, 36*(2), 2260–2265.
Gigerenzer, G. (2004). Mindless statistics. *The Journal of Socio-Economics, 33*(5), 587–606.
Gillie, T., & Broadbent, D. (1989). What makes interruptions disruptive? A study of length, similarity, and complexity. *Psychological Research, 50*(4), 243–250.
Goos, M., Galbraith, P., & Renshaw, P. (2002). Socially mediated metacognition: Creating collaborative zones of proximal development in small group problem solving. *Educational Studies in Mathematics, 49*(2), 193–223.
Guzdial, M. (2007). Contextualized computing education increasing retention by making computing relevant. *White Paper, Georgia Institute of Technology.*

Guzdial, M., & Forte, A. (2005). Design process for a non-majors computing course. *ACM SIGCSE Bulletin, 37*(1), 361–365.

Guzdial, M., Hübscher, R., Nagel, K., Newstetter, W., Puntambekar, S., Shabo, A., et al. (1997). Integrating and guiding collaboration: Lessons learned in computer-supported collaborative learning research at Georgia Tech. In R. Hall, N. Miyake, & N. Enyedy (Eds.), *Proceedings of the 2nd International Conference on Computer Support for Collaborative Learning* (pp. 91–100). Mahwah: Lawrence Erlbaum Associates, Inc.

Jermann, P., Mühlenbrock, M., & Soller, A. (2002). Designing computational models of collaborative learning interaction. In *Proceedings of the conference on computer support for collaborative learning: Foundations for a CSCL community* (pp. 730–732). Boulder: International Society of the Learning Sciences.

Kangas, M. (2004). The impact of individual differences on pair programming. *T-76.650 Seminar in Software Engineering*.

Katira, N., Williams, L., Wiebe, E., Miller, C., Balik, S., & Gehringer, E. (2004). On understanding compatibility of student pair programmers. *SIGCSE Bulletin, 36*(1), 7–11.

Katira, N., Williams, L., & Osborne, J. (2005). Towards increasing the compatibility of student pair programmers. In *Proceedings of the 27th International Conference on Software Engineering* (pp. 625–626). St. Louis: ACM.

Kelleher, C., & Pausch, R. (2007). Using storytelling to motivate programming. *Communications of the ACM, 50*(7), 58–64.

Koehler, M. J., & Mishra, P. (2005). What happens when teachers design educational technology? The development of technological pedagogical content knowledge. *Journal of Educational Computing Research, 32*(2), 131–152.

Li, Z., Plaue, C., & Kraemer, E. (2013). A spirit of camaraderie: The impact of pair programming on retention. In *Software Engineering Education and Training (CSEE&T), 2013 I.E. 26th Conference on* (pp. 209–218). IEEE.

Lyons, L., Tissenbaum, M., Berland, M., Eydt, R., Wielgus, L., & Mechtley, A. (2015). Designing visible engineering: supporting tinkering performances in museums. In *Proceedings of the 14th International Conference on Interaction Design and Children* (pp. 49–58). New York: ACM.

Margolis, J., & Fisher, A. (2003a). Geek mythology. *Bulletin of Science Technology & Society, 23*(1), 17–20. doi: 10.1177/0270467602239766.

Margolis, J., & Fisher, A. (2003b). *Unlocking the clubhouse: Women in computing*. Cambridge: The MIT Press.

Margolis, J., Goode, J., Holme, J. J., & Nao, K. (2008). *Stuck in the shallow end: Education, race, and computing*. Cambridge: The MIT Press.

Martin, T., Berland, M., Benton, T., & Smith, C. P. (2013). Learning programming with IPRO: The effects of a mobile, social programming environment. *Journal of Interactive Learning Research, 24*(3), 301–328.

McKinney, D., & Denton, L. F. (2006). Developing collaborative skills early in the CS curriculum in a laboratory environment. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education* (pp. 138–142). New York: ACM.

Mishra, P., & Koehler, M. (2006). Technological pedagogical content knowledge: A framework for teacher knowledge. *The Teachers College Record, 108*(6), 1017–1054.

Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., & Balik, S. (2003). Improving the CS1 experience with pair programming. *SIGCSE Bulletin, 35*(1), 359–362.

Papert, S. (1980). *Mindstorms: Children, computers, and powerful ideas*. New York: Basic Books.

Papert, S., & Harel, I. (1991). Situating constructionism. *Constructionism* (1–11).

Pardos, Z. A., Baker, R. S., San Pedro, M. O., Gowda, S. M., & Gowda, S. M. (2013). Affective states and state tests: Investigating how affect throughout the school year predicts end of year learning outcomes. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge* (pp. 117–124). New York: ACM.

Perlow, L. A. (1999). The time famine: Toward a sociology of work time. *Administrative Science Quarterly, 44*(1), 57–81.

Petrick, C., Berland, M., & Martin, T. (2011). Allocentrism and computational thinking. In G. Stahl, H. Spada, & N. Miyake (Eds.), *Proceedings of the Ninth International Conference on Computer-Supported Collaborative Learning*. Hong Kong.

Preston, D. (2005). Pair programming as a model of collaborative learning: A review of the research. *Journal of Computing Sciences in Colleges, 20*(4), 39–45.

Purtilo, J. J., & Callahan, J. R. (1989). Parse tree annotations. *Communications of the ACM, 32*(12), 1467–1477.

Radermacher, A., Walia, G., & Rummelt, R. (2012). Assigning student programming pairs based on their mental model consistency: an initial investigation. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (pp. 325–330). Raleigh: ACM.

Repenning, A., Ahmadi, N., Repenning, N., Ioannidou, A., Webb, D., & Marshall, K. (2011). Collective programming: Making end-user programming (more) social. In *End-user development* (pp. 325–330). Berlin: Springer.

Resnick, M., Maloney, J., Monroy-Hernández, A., Rusk, N., Eastmond, E., Brennan, K., & Millner, A. (2009). Scratch: Programming for all. *Communications of the ACM, 52*(11), 60–67.

Romero, C., & Ventura, S. (2010). Educational data mining: A review of the state of the art. *IEEE Transactions on Systems Man and Cybernetics Part C: Applications and Reviews, 40*(6), 601–618.

Roth, W.-M., Woszczyna, C., & Smith, G. (1996). Affordances and constraints of computers in science education. *Journal of Research in Science Teaching, 33*(9), 995–1017.

Salleh, N., Mendes, E., Grundy, J., & Burch, G. S. J. (2010). An empirical study of the effects of conscientiousness in pair programming using the five-factor personality model. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1* (pp. 577–586). Cape Town: ACM.

Salomon, G., & Globerson, T. (1989). When teams do not function the way they ought to. *International Journal of Educational Research, 13*(1), 89–99. doi:10.1016/0883-0355(89)90018-9.

Salton, G. (1989). *Automatic text processing*. Boston: Addison Wesley.

Sanders, D. (2002). Student perceptions of the suitability of extreme and pair programming. In M. Marschesi, G. Succi, D. Wells, & L. Williams (Eds.), *Extreme programming examined* (pp. 261–271). Boston: Addison-Wesley.

Sao Pedro, M. A., de Baker, R. S., Gobert, J. D., Montalvo, O., & Nakama, A. (2013). Leveraging machine-learned detectors of systematic inquiry behavior to estimate and predict transfer of inquiry skill. *User Modeling and User-Adapted Interaction, 23*(1), 1–39.

Schwartz, D., & Martin, T. (2004). Inventing to prepare for future learning: The hidden efficiency of encouraging original student production in statistics instruction. *Cognition and Instruction, 22*(2), 129–184.

Shaffer, D. W., & Resnick, M. (1999). "Thick" authenticity: New media and authentic learning. *Journal of Interactive Learning Research, 10*(2), 195–215.

Singer, M., Radinsky, J., & Goldman, S. R. (2008). The role of gesture in meaning construction. *Discourse Processes, 45*(4–5), 365–386.

Soller, A., Martínez, A., Jermann, P., & Muehlenbrock, M. (2005). From mirroring to guiding: A review of state of the art technology for supporting collaborative learning. *International Journal of Artificial Intelligence in Education, 15*(4), 261–290.

Soloway, E., & Spohrer, J. C. (Eds.). (1989). *Studying the novice programmer*. Hillsdale: Lawrence Erlbaum Associates.

Speier, C., Vessey, I., & Valacich, J. S. (2003). The effects of interruptions, task complexity, and information presentation on computer-supported decision-making performance. *Decision Sciences, 34*(4), 771–797. doi:10.1111/j.1540-5414.2003.02292.x.

Srikanth, H., Williams, L., Wiebe, E., Miller, C., & Balik, S. (2004). On pair rotation in the computer science course. In *Proceedings of the 17th Conference on Software Engineering Education and Training* (pp. 144–149).

Talavera, L., & Gaudioso, E. (2004). Mining student data to characterize similar behavior groups in unstructured collaboration spaces. In *Proceedings of the Artificial Intelligence in Computer Supported Collaborative Learning Workshop at the ECAI 2004*.

Teague, D. M. (2009). A people-first approach to programming. In *Proceedings of the Eleventh Australasian Conference on Computing Education - Volume 95* (Vol. Wellington, New Zealand, pp. 171–180). Darlinghurst: Australian Computer Society, Inc.

Teague, D. M., & Roe, P. (2009). Learning to program : From pear-shaped to pairs. In *International Conference on Computer Supported Education (CSEDU2 2009)* (pp. 151–158). Lisboa; INSTICC Press: The Institute for Systems and Technologies of Information, Control and Communication. Retrieved from http://eprints.qut.edu.au/29995/

Tudge, J. R. H. (1992). Processes and consequences of peer collaboration: A Vygotskian analysis. *Child Development, 63*(6), 1364–1379.

Van Toll, T., Lee, R., & Ahlswede, T. (2007). Evaluating the usefulness of pair programming in a classroom setting. In *6th IEEE/ACIS International Conference on Computer and Information Science (ICIS 2007)* (pp. 302–308).

Watkins, K. Z. B., & Watkins, M. J. (2009). Towards minimizing pair incompatibilities to help retain under-represented groups in beginning programming courses using pair programming. *Journal of Computing Sciences in Colleges, 25*(2), 221–227.

Werner, L. L., Hanks, B., & McDowell, C. (2004). Pair-programming helps female computer science students. *Journal on Educational Resources in Computing, 4*(1), 4.

Wilensky, U. (1999). NetLogo, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL. Retrieved from http://ccl.northwestern.edu/netlogo/