Designing IDE Interventions to Promote Social Interaction and Improved Programming Outcomes in Early Computing Courses

2

DANIEL OLIVARES, CHRISTOPHER HUNDHAUSEN, and NAMRATA RAY,
Human-centered Environments for Learning and Programming (HELP) Lab, School of Electrical
Engineering and Computer Science, Washington State University, Pullman, WA, USA

As in other STEM disciplines, early computing courses tend to stress individual assignments and discourage collaboration. This can lead to negative learning experiences that compel some students to give up. According to social learning theory, one way to improve students' learning experiences is to help them form and participate actively in vibrant social learning communities. Building on social learning theory, we have designed a set of software interventions (scaffolds and prompts) that leverage automatically collected learning process data to promote increased social interactions and better learning outcomes in individual programming assignments, which are a key component of early undergraduate computing courses. In an empirical study, we found that students' interaction with the interventions was correlated with increased social activity, improved attitudes toward peer learning, more closely coupled social networks, and higher performance on programming assignments. Our work contributes a theoretically motivated technological design for social programming interventions; an understanding of computing students' willingness to interact with the interventions; and insights into how students' interactions with the interventions are associated with their social behaviors, attitudes, connectedness with others in the class, and their course outcomes.

CCS Concepts: • **Human-centered computing** → **Collaborative and social computing**; **Empirical studies in collaborative and social computing**; • **Social and professional topics** → **Computer science education**; • **Applied computing** → **Interactive learning environments**; *Collaborative learning*;

Additional Key Words and Phrases: Social programming interventions, software scaffolding, integrated development environment (IDE), learning analytics, social learning theory, computing education, CS1

ACM Reference format:

Daniel Olivares, Christopher Hundhausen, and Namrata Ray. 2021. Designing IDE Interventions to Promote Social Interaction and Improved Programming Outcomes in Early Computing Courses. *ACM Trans. Comput. Educ.* 22, 1, Article 2 (October 2021), 29 pages.
<https://doi.org/10.1145/3453165>

The work was supported by the National Science Foundation under grant no. IIS-1321045.

Authors' address: D. Olivares, 502 E. Boone Ave., AD Box 26, Spokane, WA 99258-0026; email: Daniel.Olivares@wsu.edu; C. Hundhausen, PO Box 642752, Pullman, WA 99163-2752; email: hundhaus@wsu.edu; N. Ray, PO Box 644020, Pullman, WA 99164-4020; email: namrata.ray@wsu.edu.

Author current address: D. Olivares, School of Engineering & Applied Science, Gonzaga University, Spokane, WA, USA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2021 Association for Computing Machinery.

1946-6226/2021/10-ART2 \$15.00

<https://doi.org/10.1145/3453165>

1 INTRODUCTION

As in other STEM disciplines, early courses in computing tend to stress individual problem solving and discourage collaboration [29]. The prevailing sentiment, reflected in computing educators' keen interest in designing "nifty" individual computer programming assignments [54] and detecting cheating [27], is that students need to learn the required programming skills on their own.

Requiring students to work individually appears to contradict students' own perception that collaborative learning is vital to success [74]. Moreover, when students are forced to work alone, they may develop the false impression that they are the only ones struggling; they may have difficulties getting the help they need; and ultimately, they may fail to develop the self-efficacy needed to persist at computer programming [67]. As an antidote, social learning theory [7, 8, 35, 68], along with a growing body of empirical evidence [66, 70], suggests that *social interaction* is positively associated with academic success in computing degree programs. This motivates the need to promote *social* learning experiences in early computing courses. To that end, we have been exploring the potential for software-realized *interventions* (including scaffolds [26] and prompts), delivered directly within the **integrated development environment (IDE)** learners use for programming assignments, to promote increased social interactions and ultimately improve learning outcomes in early undergraduate computing courses, which have been traditionally viewed as having high failure rates [11, 85], although a recent analysis [76] suggests that their failure rates may not be inordinately high compared to other STEM disciplines.

To provide a basis for such interventions, we leverage learning process data that can be automatically collected through an IDE, where introductory students spend significant amounts of time writing, compiling, executing, and debugging the programming solutions they submit for course credit. Prior research has explored the automatic collection of *programming process data* [17, 36] within the IDE, and how to leverage such data to predict and promote future success [15, 32, 83, 84]. In addition, researchers have recently begun to explore the automatic collection of *social process* data on students' interactions within an online learning community (see [18]), and how such social process data might be leveraged to improve students' social interactions and learning.

Building on this foundation, this article explores the following research questions:

- RQ1:** In early computing courses, to what extent will students interact with software-realized interventions designed to steer them toward increased social interaction within an online learning community?
- RQ2:** Are students' preexisting attitudes correlated with the extent to which they interact with the interventions?
- RQ3:** Are students' levels of interaction with the interventions associated with increased social and programming behaviors, improved attitudes toward learning, and higher learning outcomes?

In addressing these questions, this article contributes (a) a theoretically- and empirically-driven technological design for software-realized interventions to promote social interaction within the context of programming assignments; (b) a new understanding of computing students' willingness to interact with the interventions; and (c) insights into how students' interactions with the interventions are associated with their social behaviors, attitudes, connectedness with others in the class, and ultimately their course outcomes.

Table 1. Principles Guiding the Design of the Software Interventions Explored in this Research

#	Principle	References
1	Interventions should guide learners toward formulating better questions, answers, and reflections	[82, 86]
2	Interventions should help learners interact with and be aware of their peers to foster a supportive learning community	[7, 9]
3	Interventions should prompt learners to reflect on their own learning process and progress	[35]
4	Interventions should furnish actionable guidance relevant to the learner's immediate context and goals	[34, 51]

2 RELATED WORK

This work builds both on foundational theory and three broad strands of education research. The first focuses on developing and leveraging socially oriented pedagogies to promote learning and retention. The second focuses on leveraging dynamically gathered learning process data to tailor and customize learning supports. The third focuses on understanding students' social help seeking behaviors and designing help systems to support them. Below, we briefly review these lines of work.

2.1 Foundational Theory and Principles

Our design of software interventions to engage students more socially in individual programming assignments builds on two social learning theories, along with principles from human-computer interaction. First, Bandura's Social Cognitive and Self-efficacy Theory [8, 9] posits that learners' social behavior and cognitive processes are affected by actions that they observe in others in their community. In the learning process, learners observe their peers and potentially act based on their perceived experiences (observational learning). They will learn from each other both from observing, imitating others ("vicarious" experiences), and from having opportunities to model their own behavior accordingly ("enactive" experiences).

Second, Vygotsky's **Zone of Proximal Development (ZPD)** Theory [82] emphasizes the gap between a learner's current independent problem-solving ability and their potential ability with the aid of guidance and/or peer collaboration. When learners are within the ZPD, interaction with more skilled instructors or peers is seen as an effective way of helping develop skills and strategies. Vygotsky proposed the concept of *scaffolding* as one way to elevate learners' current problem-solving ability [86]. Scaffolding aims to provide learners with resources to get them past periods where they may become stuck and frustrated. This is usually done through some type of minimal support, including partial solutions, directions, or guidance (see. e.g., [41]). More recently, *software-realized scaffolding* [26], on which this work builds directly, has been explored as a way to assist learners by integrating coaching, prompts and other learning aids directly into computer-based learning environments.

Third, we draw on principles from human-computer interaction [34, 51] to guide the design of the user interface for the software interventions. While several principles are relevant to the design of the interventions, a key principle is that interventions should present actionable guidance relevant to learners' immediate context and tasks.

Table 1 presents four guiding principles derived from the above foundations. As will be shown in Section 3.2, these principles motivate and guide the design of the software interventions explored in this research.

2.2 Social Pedagogies and Supporting Technologies

Our work is rooted in pedagogical approaches that situate learning around social interaction. Several such pedagogical approaches have been explored in computing education, including Problem-Based Learning [52], **Studio-Based Learning (SBL)**, see [16] for a review), **Pair Programming (PP)**, see [42, 62]) and **Peer Learning (PL)**, see [59, 60]).

Computing education researchers have developed various educational technologies both to support these pedagogies and to make them more practical and convenient to implement. Throughout this line of work, there has been an emphasis on online learning environments and **integrated development environments (IDEs)**, where computing students spend significant amounts of time developing code solutions. For instance, a number of IDEs [24, 33, 73] support real-time co-editing of program code in a web browser. However, because these tools focus on allowing multiple programmers to program collaboratively (as in PP), they are not appropriate for the social programming contexts studied of interest to our work, in which students interact as they work on their own individual solutions.

HelpMeOut [28] and Crowd::Debug [44] build on a social community in a different manner: through a social recommendation system based on possible solutions to programming errors gleaned from IDE data collected on a community of programmers. In evaluation studies, novice programmers found useful fixes for up to 57% of errors. These tools differ from this work in that, while they draw from the actions of a community of programmers, they do not support or encourage social interaction among community members.

Another type of educational technology aims to promote social interaction through *peer review*, a form of SBL. For example, OSBLE facilitates both face-to-face and online “design crits” of computer code solutions [30]—a key activity within SBL. In a similar vein, a number of tools have been developed to support peer reviews in which students are tasked with assessing the code solutions of their peers [21, 22, 31, 58, 64]. This line of work shares our interest in engaging learners in social learning processes. However, whereas this line of work mandates such social learning processes, our work instead seeks opportunities to guide students toward greater social engagement during individual programming assignments.

A final prominent example of an educational technology designed to promote social interaction is Scratch [12, 40], which focuses on building a development and learning community through collaboration and the sharing of code. While our work shares Scratch’s interest in building a community around programming practices, it differs in two key respects: (a) it focuses on undergraduate computer science education instead of K-12 education, and (b) it focuses on providing learner-customized interventions to encourage social behaviors within a learning community during the programming process.

2.3 Leveraging Learning Process Data to Provide Guidance

Our interest in providing students with dynamic guidance based on learning process data has its roots in prior research on *intelligent tutoring systems* and *learning analytics*. **Intelligent tutoring systems (ITSs)** provide learners with automated guidance on learning tasks. They do this by drawing inferences from student learning processes, using those inferences to model student learning so that guidance can be dynamically adapted to students’ needs within a known problem space [39]. Several ITSs have been built to support the learning of computer programming. Perhaps the earliest example is the Lisp tutor [5], a so-called cognitive tutor that helped students complete programming problems in Lisp. In a similar vein, Butz et al. [13] developed an interactive, adaptive ITS to help computer science students navigate course material. For comprehensive reviews of the educational effectiveness of ITSs in computing education, see the meta-study of Nesbit et al. [45].

Our work takes an approach similar to that of ITSs. It continuously collects learner data and, based on the context of those data, generates interventions designed to influence student behavior. Unlike ITSs, however, we do not possess an exact solution to a given programming problem, and hence are not aware of a correct solution path. Moreover, unlike ITSs, which rely on individual learning process data within a constrained problem-solving environment, our work focuses on *social* data in an unconstrained problem-solving environment (an IDE).

Learning analytics [79–81] denotes the process of collecting and analyzing information about students’ learning processes and progress in order to provide an empirical foundation on which adjustments to teaching and learning can be made. Learning analytics dashboards attempt to help learners and instructors make sense of the abundance of learner data that are increasingly available [80]. These dashboards can be used to augment face-to-face teaching, online learning, or even blended learning settings [79]. They might be integrated into general-purpose learning management systems (e.g., [66]) or be part of student-centered dashboards that make use of both learning analytics and formative assessment [4]. In computing education, a prominent example is Classroom Salon [10], which provides an environment in which programming students can collaborate in editing and commenting on code. Though code cannot be run or compiled in this environment, it does provide a social environment in which computer science students can cooperatively create, comment on, and modify documents. For instructors, the dashboard provides charts to help visualize statistical information about students’ document annotations. Though it draws heavily on learning analytics dashboards as utilized in Classroom Salon, our work differs in that we integrate dashboards into a combined social and programming environment, thus allowing the interventions to be based on both social and programming behaviors.

2.4 Help Seeking

Our work is influenced by a body of research on learners’ social help-seeking behaviors, a crucial component of *self-regulation* during the learning process [47]. In computing education, it has been observed that students often avoid seeking help from others [20]. Their willingness to seek help may be influenced by many factors, including demographics [71], attitudes [72], motivation [72], and perceived social consequences [70].

Regardless of learners’ predispositions toward help seeking, researchers have been interested in studying learners’ help-seeking strategies [78]; developing general models of effective help seeking [46], and ultimately integrating help-seeking supports into computer-based learning environments [3]. For instance, in the ITS work described above, researchers have explored *adaptive* help systems that tailor help content to learners based on their past performance [87]. An alternative approach is to provide automated feedback on students’ help-seeking actions during the learning process [65]. Our study builds on this work by designing software interventions to encourage and ease social help seeking within the context of programming assignments in early computing courses.

3 SOFTWARE INTERVENTION DESIGN

This work explores the value of software interventions in promoting educationally beneficial social behaviors during individual programming assignments and ultimately improving student learning outcomes. To increase the chances that students see the interventions when they work on programming assignments, we embedded them in a “social” IDE—that is, an IDE augmented with a social network style activity stream, as developed in our prior work [15, 17].

Figure 1 presents a screenshot of the social IDE, which augments a commercial IDE (Microsoft Visual Studio) with features to promote and facilitate social interaction. An embedded social network style activity feed, in which students can discuss the programming assignment they are working on, spans the right-hand side of the IDE. Students can ask for help by right-clicking on

Fig. 1. IDE augmented with social features.

a block of code or a programming error and choosing “Ask for Help” from a context menu. This leads to the highlighted code block or error being automatically inserted into an activity feed post, which the user can further edit before posting.

To further increase students’ potential exposure to the interventions, we also embedded them into OSBLE+ [53], the online **learning management system (LMS)** used by students in the courses in which we conducted this research. We embedded the interventions into the OSBLE+

dashboard (Figure 2), which includes an activity feed that mirrors the activity feed embedded within the Social IDE (see bottom center of Figure 2).

3.1 Design Process

In this research, we augmented the social IDE and OSBLE+ activity feed with software prompts and scaffolds [26] designed to steer students toward educationally beneficial social and programming behaviors. As a starting point for our design, we drew on our foundational social learning theories (see Section 2.1), which hold that learning can be facilitated through a social learning community in which learners seek help from others, give help to others, and reflect on their progress in a public forum. The theories suggest three types of interventions that, at strategic points in the learning process, (1) prompt students to ask for help and provide guidance on how to do so; (2) prompt students to give help to others in need; and (3) prompt students to talk about learning content and self-reflect on their learning process and progress.

To converge on the specific design of these interventions, we completed three iterations of a user-centered design process [25, 50]. In the first iteration, we presented students with mockups of early prototype designs of potential interventions and asked them to provide written feedback on various aspects of the design. In the second two iterations, we designed low fidelity prototypes of the interventions based on feedback from the previous iteration, and then conducted evaluation studies in which students worked through sample scenarios with those prototypes. A total of 21 students recruited from early computing courses (CS1 and CS2) at our home institution participated in the studies: 10 in the first iteration, 5 in the second iteration, and 6 in the third iteration.

3.2 Final Design

Below, we present the final design of the software interventions that came out of the iterative design process; how they are integrated into our social IDE; and our design rationale. A more detailed account of our design process, including intermediate designs and evaluation study findings, can be found in [53].

3.2.1 Overview. Our iterative design process converged on eight different interventions to prompt for and guide three forms of educationally beneficial behavior: help-seeking, help-giving, and social interaction. Table 2 presents a summary of each type of intervention, including a brief description of the behavior it prompts for, the specific conditions that trigger it, and its theoretical rationale, which references the principles presented in Table 1. As the table indicates, help-seeking interventions, which aim to encourage students to reach out for help from others, are strategically presented both when students encounter compilation and run-time errors and when others declare themselves available for help. Help-giving interventions, which are designed to encourage students to give help to others, are presented when students submit a completed assignment at least one day ahead of time or go for 10 minutes (which may include periods of time away from the computer) without encountering compilation or run-time errors. Finally, social interaction interventions, which aim to engage students in discussions about course content and reflections about their own programming activities, appear when students go for a day without making a post or reply to the activity feed, or when students submit an assignment. To avoid overwhelming the user, we require a minimum of 10 minutes to elapse (which may include periods of time away from the computer) between any two interventions of the same type.

3.2.2 Intervention Access and Functionality. The above interventions are made available in two locations:

Fig. 2. LMS dashboard with activity feed.

Table 2. Summary of Software Interventions

Category	Prompt	Trigger	Theoretical Rationale (See Principles in Table 1)
Help-seeking	1. Ask for help on compilation error from others, with suggestions for whom to ask. 2. Ask for help on run-time error from others, with suggestions for whom to ask. 3. Ask for help from those who are available to help.	1. Student obtains five compilation errors in last 10 minutes. 2. Student obtains five run-time errors in last 10 minutes 3. Another student has just made themselves available for help	1, 2, 4
Help-giving	1. Consider helping students who have asked for help. 2. Consider helping students who have asked for help.	1. Student completes 10 minutes of programming activity without a compilation or build error. 2. Student submits an assignment at least one day before submission deadline.	1, 2, 4
Social interaction	1. View and change availability status. 2. View posts on popular topics or make a new post 3. Make a post in which you reflect on your programming process	1. Student has clicked on “View/Change” status link. 2. Student has not posted/replied to activity feed for one day. 3. Student submits an assignment	1, 2, 3

- Within the Activity Feed panel of the Social IDE (see Figure 1), interventions appear in a separate “Suggestions” pane below the Activity Feed (see Figure 3).
- Within the Activity Feed panel of the OSBLE+ Dashboard (see Figure 2), interventions appear between the search bar and the actual posts (see Figure 4).

In the Social IDE, the “Suggestions” pane is docked below the Activity Feed pane by default. At the top of this pane is a list of icons that enable users to access to the following features relevant to the “Suggestions” and the online social environment to which they pertain (refer to numbered labels in Figure 5):

1. *New suggestion notification icon.* Conveys when a new suggestion is available. The entire “Suggestions Dashboard” header also blinks for 15 seconds when a new suggestion is present.
2. *View private conversations.* Show listing of all private conversations between the current user and other users.
3. *View status/availability.* Show who is available for help and set own availability status.
4. *View dismissed suggestions.* Show previous suggestions that were closed and are no longer in view.
5. *View profile page.* Show listing of user’s posts and replies.
6. *Provide feedback.* Provide open-ended comments and feedback on the “Suggestions” system.
7. *View Help.* Open online help page in web browser.

Likewise, in the Activity Feed panel of the LMS Dashboard, this same functionality is available through a set of icons that appear at the top of the Suggestions panel (see Figure 5).

Fig. 3. Integration of interventions into “Suggestions” pane of Social IDE (lower right).

Fig. 4. Integration of interventions into OSBLE+ Activity Feed.

Fig. 5. “Suggestions” Dashboard provides access to features that support use of interventions.

Fig. 6. Help-Seeking Prompts.

Each time an intervention is generated in response to a trigger event (see Table 2), it is injected at the top of the “Suggestions” pane. Note that only one intervention of each type can be active (i.e., not “dismissed”) at a time. Any time a new intervention is presented when an intervention of the same type is still active, the older intervention will be automatically dismissed on the assumption that it is no longer relevant. This ensures that each active intervention will be the most relevant one to the user.

3.2.3 Help-Seeking Interventions. Figure 6 shows the three prompts designed to encourage help-seeking. As the figure suggests, the “Build Errors” and “Runtime Errors” prompts reference the specific errors encountered by the user, while the “Others Available to Help” prompt references specific peers who have made themselves available for help. Clicking on these prompts opens a structured form designed to scaffold the formulation of each type of question. For example, Figure 7 shows the structured form a student could fill out to ask for help on a specific run-time error encountered. Notably, the structured forms enable a student to post a question *anonymously*, thus removing a potential barrier to posting the question.

3.2.4 Help-Giving Interventions. Figure 8 presents the two interventions designed to encourage help-giving. The first intervention, which is triggered when a student completes 10 minutes of

Ask a Question!

Current Course: Was this suggestion helpful?

Your program ran into a run-time error. *Other classmates' programs also had run-time errors recently.*
Would you like to ask your classmates about this issue?

Ask the question: Clear Template Text

Hey all, I recently got a run-time error:
[Win32 Exceptions: 0xC0000005: "Exception thrown at 0x00EF4E48 in ConsoleApplication3.exe: 0xC0000005: Access violation reading location 0x2D6ADB7C." on Line 10 Near: " int b = i / balance[6000000000];"]

Does anyone have any tips on how I could resolve this?

I have tried [... describe what you've tried so far...]

Here's part of my code: [\(Edit\)](#)

```

/***** ConsoleApplication3: main.c *****/
/* my hello world C */

printf("Hello, World! \n");
int i;
double balance[5] = { 1000.0, 2.0, 3.4, 7.0, 50.0 };
int b = i / balance[6000000000];
scanf("%d", &i);
return 0;
}

/***** end code excerpt *****/
    
```

View Classmates' Questions Post Question to Class (Anonymously) Post Question to Class

Fig. 7. Structured form to help students ask a question related to a run-time error.

Help other Students! X

Other students have encountered issues you may have recently resolved. << Help your classmates out! >>

Help Your Classmates! X

You've submitted your assignment early! << Let your classmates know you can help them! >>

Fig. 8. Help-giving prompts.

programming activity without an error, encourages students to answer questions that have not yet been resolved. As illustrated in Figure 9, clicking on this intervention brings up a structured form that guides students either to reply to specific unanswered questions or to mark existing replies to those questions as helpful, thus giving those replies increased credibility. The second intervention, which is triggered when a student submits an assignment at least one day before the deadline, provides a template that guides students to share their reflections on their programming process (see Figure 10). Additionally, students who receive this intervention are encouraged to update their availability to indicate that they are now available to help others.

3.2.5 Social Interaction Interventions. Figure 11 presents three interventions designed to encourage students to interact with others in the course. The first intervention (see Figure 12) is

Fig. 9. Structured form to help students to reply to others' questions.

triggered when a student clicks on the “View/Change” status icon (see Figure 4 and Figure 5). It presents an overview of who is currently available to give help, as well as their window of availability. The intervention also provides options to change the student’s status and availability and to ask a new question directed at either the entire class or a specific set of students. The second intervention (see Figure 13), which is triggered when a student has not posted to the activity feed in the past day, provides a list of topic hashtags most recently used in the activity feed. By clicking on those hashtags, the student can explore posts related to those hashtags. In addition, the intervention encourages the student make a post on a new topic and to create a hashtag for that topic. The third intervention is triggered when a student submits an assignment within a day of the due date (i.e., not “early” as we have defined it). Analogous to the intervention shown in Figure 10, the intervention provides a structured form that encourages students to reflect on their programming process.

4 EMPIRICAL EVALUATION

To evaluate the effectiveness of the above software interventions in promoting positive changes in students’ social and programming behaviors, we conducted a series of mixed-methods empirical studies.

Below, we present the final study in that series, which addresses the three research questions posed in Section 1. For a full account of all our empirical studies, please see [53].

4.1 Method

4.1.1 Design. Our original empirical evaluation was designed as a within-subjects quasi-experimental study [61, 75] in a CS1 course at our home institution. As they completed programming assignments, students in the study were exposed during the first half of the course to a Control treatment in which they did not have access to the interventions. During the second half of the course, students were exposed to an Experimental treatment in which they did have access to the interventions. Table 3 presents a week-by-week breakdown of course topics included in each treatment.

As it turned out, students in the study varied considerably in the extent to which they interacted with the interventions. Some students’ lack of engagement with the intervention created gaps

Help out your classmates!

Current Course:

Was this suggestion helpful?

You have successfully submitted your assignment early! *Some of your classmates' programs have encountered errors...*

Would you like to offer your help?

Let your classmates know:

Clear Template Text

Hey all, I'm done with test PA and am available if anyone wants to talk about it!

I found [...] aspect of assignment [...] especially [interesting/fun/tricky/etc.].

I overcame these issues by [...] describe how you resolved these issues ...]

One thing I learned that that might help you is: [...] describe what you learned while completing this assignment ...]

Update Your Status

Status Message:

Working on PA3

Availability:

You're available

Until:

July 2017

@ 9:30 PM

Su	Mo	Tu	We	Th	Fr	Sa
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

(Updating your status will let other users know that you are open to answer questions related to the current course assignments and activities)

View Classmates' Questions

ONLY Update Status

Post to Class AND Update Status

Fig. 10. Structured form to help students reflect on programming process.

You Are Available!

You have said you're available to help your classmates. << View/Change your status. >>

Make A Post!

What's going on right now? << Tell others what you're doing! >>

Make a Post!

You've submitted your assignment. << Make a post about it! >>

Fig. 11. Social interaction prompts.

in the experimental treatment data, leading to inconclusive results regarding the impact of the interventions.

Given the inconclusive results of our original study, we decided to modify the design: Rather than trying to assert, through a controlled experiment, a *causal* link between students' interac-

Availability Details

Current Course: [dropdown]

Was this suggestion helpful? [thumbs up/down]

Other users currently available:

- ☒ Select All
- ☒ Bob Smith: "Office Hours " until 7/25/2017 1:30 AM
- ☒ Carol Jackson: "Working on PA3 " until 7/24/2017 9:30 PM

Modify Your Status

Status Message:
Working on PA3

Availability:
☒ You're available

Until:
[Calendar: July 2017, 24th highlighted] [Time: 9:30 PM]

(Updating your status will let other users know that you are open to answer questions related to the current course assignments and activities)

Update Status

Ask these users a question:

Find out what people are talking about. Ask these classmates or the entire class a question!

[Post Question to Class (Anonymously)] [Post Question to Class] [Post Question to Selected Classmates]

Fig. 12. Structured form to update availability status and post a question.

tion with the interventions and their social behaviors, social connections, attitudes, and academic performance, we performed a follow-up study to identify possible *correlations* between students' level of interaction with the interventions in the final four weeks of the course and the outcome variables. Below, we present the follow-up study.

4.1.2 Participants. Participants included 41 students (33 male, 8 female, age range 18 to 30 y.o., $M = 22$ y.o.) enrolled in two offerings of the CS1 course at our home institution, a large research university in the western U.S. These included all 26 students enrolled in the summer 2017 offering, along with 15 of the 19 students enrolled in the summer 2018 offering. Participants were a mix of computer science majors (46%) and non-computer science majors (54%) from engineering,

Fig. 13. Structured gorm to prompt dtudent to rxplore rexisting discussions and/or make post on new topic.

Table 3. Course Topics by Week and Treatment

Week #	Topic	Treatment
1	Software & C Language Introduction, Arithmetic, Functions	Control
2	Functions, File Processing (IO), Conditionals	
3	Conditionals, Loops	
4	Modular Function Design	
5	Arrays, Data Types	Experimental
6	C Strings, Structs	
7	Recursion, Dynamic Memory	
8	Binary & Bit operations, C Macros	

mathematics, and business fields. The first author served as the course instructor for both course offerings.

4.1.3 Courses and Materials. The two courses involved in the study were nearly identical. Focusing on computer programming in the C language, each course condensed the same material normally covered in a sixteen-week semester into an eight-week period. Moreover, each course had the same instructor, textbooks, labs, assignments, and exams.

Students in the courses had access to a nearly identical set of course materials, all of which were available online through the course LMS. These materials consisted of handouts, prompts for 12 programming labs, prompts and grading rubrics for eight individual programming assignments, weekly code samples, weekly lecture slides, weekly quizzes, and a midterm and final exam.

Each course adopted an identical plagiarism and academic integrity policy. High-level collaboration was encouraged, but code copying and pair programming were not allowed. To ensure that students’ code submissions were original, the instructor submitted students’ submissions to the

Table 4. Detailed Description of Log Data Collected

Data Category	Event	Description
Social Activities	Ask-For-Help	Made top-level post from the Social IDE
	Hashtag	Clicked on link topic tag in posts/replies, e.g., #PA1Help
	Initiate Reply	Clicked on the “Reply” button in activity feed
	Keyword Search	Used the “Filters & Search” feature on the activity feed
	Mention	Clicked an @Mention user tag, e.g., @BobSmith
	Post	Made top-level post from the LMS
	Post Details	Clicked on the “Details” link in a top-level post
	Reply Marked Helpful	Clicked on the “Thumbs up” in a post reply
	Reply	Posted reply to a top-level post
Programming Activities	View Replies	Clicked the “View Replies (#)” link in a top-level post
	Build	Compiled code project
	Cut-Copy-Paste	Used cut, copy, or paste operation in the Social IDE code editor
	Debug	Added debugging breakpoint in Social IDE
	Editor	Edited code files in the Social IDE within a 1-minute window
	Build Error	Obtained compilation error(s) in Social IDE
	Exception	Obtained runtime exceptions in Social IDE
	Save	Saved a code file within the Social IDE
	Submit	Submitted assignment solution through Social IDE

MOSS plagiarism detection software [1]. All exams were administered in-class under the instructor’s supervision.

Two additional course materials were central to this study. First, during the final four weeks of each course, students used a *Social IDE* and *LMS* that included the interventions described in Section 3. Second, *online surveys* were used to gauge student attitudes before and after students’ exposure to the interventions. The surveys consisted of questions from the **MSLQ: Motivated Strategies for Learning Questionnaire** [57], C++ Self-Efficacy [63], Classroom Community scales [69], and the Sociability Scale [37].

4.1.4 Data Collected. We gathered four types of evaluation data on participants in this study:

8. *Demographic data*, including participants’ gender, age and academic major.
9. *Attitudinal data* through online surveys administered before and after exposure to the interventions.
10. *Log data* on participants’ programming processes and social activities within the Social IDE and the LMS (see Table 4).
11. *Learning outcomes data*: students’ grades for programming assignments, course participation, labs, exams and the overall course.

4.1.5 Procedure. At the beginning of the two courses in which this study took place, the course instructor briefly advertised the study and invited students to participate. The advertisement emphasized that participation in the study would not affect students’ course activities; the only

Fig. 14. Histogram of total intervention interactions by number of students.

difference was that students who consented to participate would release their data for research purposes. Those who chose to participate completed a written informed consent form, which also included basic demographic questions. The study protocol and informed consent form were approved by our Institutional Review Board.

Prior to their exposure to the interventions, and again at the end of the course, students were required to complete the same online attitudinal survey to gauge changes in their attitudes.

During the final four weeks of the course, students were required to complete individual programming assignments using the Social IDE and LMS. To incentivize online social participation during the programming assignments, the course instructor established a requirement of two posts and replies per week.

Students' posts and replies were not graded on quality. However, any posts or replies that did not contain academic or social content (e.g., "Post 1: Obligatory post") were removed from the analysis.

At the end of the course, the survey data, log data and grades data of those students who consented to participate in the study were collected for analysis.

4.2 Results

To explore the extent to which students interacted with the interventions (RQ 1), Figure 14 presents a histogram of the frequency of each number of total interactions. On average, students interacted with the interventions 12.83 times ($sd = 9.27$) during the four-week period in which the interventions were available. As indicated by the figure and reflected in the high standard deviation, students' level of interaction with the interventions varied considerably, with five students having fewer than five interactions and another five students having more than 25 interactions.

To shed further light on students' interaction with the interventions, Table 5 presents students' level of interaction with each of the eight interventions described in Section 3. For each interaction type, the table shows the average number of interventions generated; (b) of those generated, the average percentage interacted with at least once; and (c) the average number of times the intervention was interacted with, given that any intervention could be interacted with more than once.

As Table 5 indicates, students did not interact at all with the interventions that were generated in response to programming errors. In contrast, interventions that prompted students to reach out to others who said they were available, or that asked students to view or change their availability status, elicited the highest levels of interaction. In the middle were interventions that encouraged

Table 5. Mean Interventions Generated and Interacted with by Software Intervention Type

Intervention	# Generated	% Interacted with*	Total Interactions**
Get Help: Runtime errors	24.41 (26.28)	0.00% (0.00%)	0.00 (0.00)
Get Help: Compile errors	228.34 (243.32)	0.00% (0%)	0.00 (0.00)
Get Help: Others available	4.95 (2.77)	47.90% (38.90%)	2.89 (2.54)
Give Help: No recent errors	3.98 (0.42)	4.27% (12.38%)	0.11 (0.40)
Give Help: Submitted assignment early	5.46 (7.58)	3.41% (16.67%)	0.17 (0.51)
Social interaction: View/change status	0.76 (0.49)	68.29% (47.11%)	4.34 (3.39)
Social interaction: Make topical post	3.78 (0.42)	6.50% (16.19%)	0.29 (0.67)
Social interaction: Make reflection post	30.41 (13.17)	2.85% (3.54%)	1.03 (0.67)

*Of the interventions generated of this type, the percentage that a student interacted with *at least once* on average.

**Number of times on average a student interacted with interventions of this type (a single intervention could be interacted with multiple times).

Table 6. Correlations between Intervention Interaction and Pre-Course Attitudes

Attitudinal Scale/Measure	r_s	p^*
Coding Self-Efficacy	-.072	.665
CCS: Connectedness	.003	.983
CCS: Learning	.104	.519
CCS: Total	.040	.802
MSLQ: Self-Learning	-.039	.807
MSLQ Peer-Learning	-.040	.803
Self-Sociability	.293	.066

*Two-tailed, $N = 41$.

self-reflective and topical posts elicited interaction; these were interacted with between three and seven percent of the time.

In the remainder of this section, we use the nonparametric Spearman's rho (r_s) to test for correlations relevant to RQ 2 and RQ 3. Based on conventions in social science and educational research, we set the threshold value for statistical significance to $p < 0.05$. Since we are testing for correlations between a single variable and up to seven others, we need to consider whether to guard against type I error (e.g., using Bonferroni correction). Given that we (a) have a small sample size, (b) are performing a small number of planned comparisons, (c) are performing nonparametric tests in sequence, and (d) are most interested in the results of the individual tests (as opposed to a single omnibus test), we take Armstrong's [6] and Perneger's [55] advice *not* to adjust for type I error. In addition, we interpret r_s values of below 0.3 as *weak* correlations, r_s values between 0.3 and 0.6 as *moderate* correlations, and r_s values above 0.6 as *strong* correlations [2].

To examine the association between students' pre-intervention attitudes and their level of interaction with the interventions (RQ 2), Table 6 presents correlations between students' pre-attitudes and interaction level. As Table 6 indicates, no significant correlations were found between students' attitudes at the start of the course and their interaction level. It is notable, however, that the correlation with self-sociability was over seven times stronger than with any other pre-intervention attitudinal measure. Thus, while students' preexisting attitudes toward self-efficacy, community

Table 7. Correlations between Intervention Interaction and Online Social and Programming Events

Event Type	r_s	p^*
Posting Events	.529	<.001
Browsing Events	.569	<.001
Programming Events	.170	.288

*Two-tailed, $N = 41$.

Table 8. Correlations between Intervention Interaction and Post-Course Attitudes

Attitudinal Scale/Measure	r_s	p^*
Coding Self-Efficacy	.094	.557
CCS: Connectedness	.092	.566
CCS: Learning	-.055	.732
CCS: Total	.080	.621
MSLQ: Self-Learning	.003	.987
MSLQ: Peer-Learning	.336	.032
Self-Sociability	.195	.221

*Two-tailed, $N = 41$.

connectedness, self- and peer-learning appear not to be predictive of the extent to which they interacted with the interventions, there is a hint in the data that students' preexisting attitudes toward self-sociability may be related to their level of interaction with the interventions.

To explore whether students' level of interaction with the interventions might be associated with increased social and programming behaviors, improved attitudes toward learning, and increased learning outcomes (RQ 3), we conducted a further series of correlational analyses. Table 7 presents correlations between students' interactions with the interventions and their (a) *posting activities*—counts of feed posts, replies and helpful marks given to replies, (b) *browsing activities*—counts of events directed toward browsing activity feed content (searches, viewing of post replies and post details); and (c) *programming activities*—build, clipboard, debug, code editing, build error, save, and assignment submission events that took place within the IDE. As the table shows, there are statistically significant correlations between intervention interactions and both types of social activities, but not between intervention interactions and programming activities. The significant correlations are moderate in size. This finding provides evidence that interaction with the interventions was positively associated with the social behaviors they were designed to promote.

Table 8 examines correlations between students' interactions with the interventions and their attitudes at the end of the course. These results identify a statistically significant correlation between students' level of interaction with the interventions and their end-of-course attitudes toward peer learning. The strength of this correlation falls in the weak range. This suggests that students who interacted with the interventions more extensively showed an increased willingness to enlist their peers in the learning process.

Table 9 considers correlations between students' interactions with the interventions and their academic course performance, as gauged by key course assessments. As can be seen, a significant correlation exists between students' intervention interactions and their programming assignment grades. The size of the correlation is moderate. This finding suggests that interaction with the interventions is associated with positive performance in the course programming assignments.

Finally, to investigate how interaction with the interventions might relate to students' social connectedness with their peers, we analyzed students' post-reply relationships as social network graphs using Gephi, a **social network analysis (SNA)** tool [23]. Table 10 presents correlations

Table 9. Correlations between Intervention Interaction and Course Grades

Course Assessment	r_s	p^*
Final Course Grade	.263	.097
Course Participation	.209	.190
Quizzes	.256	.107
Programming Assignments	.355	.023
Lab Final Exam	.089	.581
Written Final Exam	.202	.205

*Two-tailed, $N = 41$.

Table 10. Correlations between Intervention Interaction and Social Network Analysis (SNA) Metrics

SNA Metric	r_s	p^*
Degree	.307	.054
Weighted Degree	.490	.001
Closeness	.174	.282
Betweenness	.095	.559
Eigenvector Centrality	.359	.023

*Two-tailed, $N = 41$.

between students' intervention interactions and five key SNA metrics: (weighted) degree, closeness, betweenness, and eigenvector centrality. Taken together, these metrics provide a sense of *social centrality* by indicating the number of connections around a specific social participant [14]. As shown in Table 10, statistically significant correlations exist between students' intervention interactions and both the weighted degree and eigenvector centrality metrics. Both correlations are moderate in strength. This finding indicates that interaction with the interventions was positively associated with the formation of stronger, more closely coupled social networks.

4.3 Discussion

Based on social learning theory and driven by log data automatically collected through the IDE, we designed the interventions presented in Section 3 to foster increased social interaction by encouraging students to ask questions, answer their peers' questions, and engage in self-reflection within the context of individual programming assignments. In turn, we predicted that such increased social interaction would lead to improved learning outcomes. While large inconsistencies in students' engagement with the interventions rendered our original quasi-experimental results inconclusive, the follow-up correlational study presented here gained useful insights into the three research questions posed in Section 1. Below, we revisit these research questions in light of our results.

4.3.1 RQ 1: To What Extent Will Students Interact with Interventions? On average, students interacted with the interventions about 12 times during the four weeks they were available. This comes out to about three times per weekly programming assignment. However, as Figure 14 reveals, students exhibited a large variance in their interaction with the interventions; indeed, the standard deviation was 9.27.

Further exploration of students' interaction with specific intervention types revealed three broad levels of interaction. First, students did not interact at all with the interventions that were generated in response to programming errors. This could have at least three possible explanations:

(1) there were usability problems with those interventions (e.g., they lacked relevance or visibility); (2) students did not feel comfortable asking their peers for help on specific error messages; or (3) students regarded other help-seeking approaches (e.g., searching for help online) as easier or more effective.

Second, interventions that encouraged self-reflective and topical posts elicited modest interaction, with interaction rates of three to seven percent. One possible explanation for this is that the prompts came at an opportune time for students—when they were done with an assignment or were in a productive period of programming. However, making reflection posts required a level of effort that many students were unwilling to make. If students had perceived the effort of performing a reflection post as worthwhile—e.g., by making such reflection a course requirement, or by making the case that engaging in and sharing one's self-reflections are valuable activities in and of themselves—then perhaps students would have been more inclined to respond to these interventions.

Finally, interventions that prompted students to reach out to others who said they were available, or that asked students to make themselves available for help, elicited the highest levels of interaction (48-68% response rates). We can infer from this data that students are more comfortable reaching out to students who said they are available, and, conversely, that students who are informed of a need for help are willing to make themselves available. We are encouraged by this finding, which suggests that struggling students in early computing courses can be motivated to reach out for help if they know others are willing to help, and that students can be motivated to help others if they know there is a need. Such help-seeking and help-giving behaviors are pillars of vibrant social learning communities.

4.3.2 RQ 2: Are Students' Preexisting Attitudes Correlated with Their Intervention Interaction?

Prior to the intervention, our study elicited students' attitudes related to self-efficacy [63], sense of community [69], self- and peer-learning, and self-sociability. These attitudinal variables were carefully selected based on their perceived relevance to students' choice to engage with the interventions. While our study was unable to identify any significant associations between these variables and students' level of interaction with the interventions, the fact that the correlation with self-sociability was found to be over seven times greater than the correlation with the other variables was notable, suggesting that further study of the relationship between self-sociability and intervention interaction may be warranted.

In addition, given the wide variance observed in students' interaction with the interventions, it would be helpful to identify additional variables that might predict students' level of interaction. Indeed, knowing what factors might influence students' choice to respond to the interventions could help better tailor the interventions for wider student use. This is clearly an area for future research.

4.3.3 RQ 3: Is Students' Intervention Interaction Associated with Increased Social Behaviors, Improved Attitudes Toward Learning, and Higher Learning Outcomes? In response to the core research question investigated by our study, we performed a series of correlational tests to identify associations between students' usage of the interventions and their social behaviors, programming behaviors, post-intervention attitudes, academic achievement, and connectedness to their peers. These tests revealed that students' level of interaction with the interventions was significantly correlated with five items:

1. Their posting events in the activity feed (strength: moderate)
2. Their browsing events in the activity feed (strength: moderate)
3. Their post-intervention attitude toward peer learning (strength: weak)

4. Their performance on course programming assignments (strength: moderate)
5. Their connectedness with peers based on the Weighted Degree and Eigenvector Centrality metrics from Social Network Analysis (strength: moderate)

Recall that, based on the social learning theory that guided this research, we designed the interventions to promote increased social activity, hypothesizing that students' programming performance would improve if they became more socially active by asking questions, answering questions, and self-reflecting on the programming process. The five significant positive correlations of weak to moderate strength identified in this research appear to provide limited support for the design goals and learning theory that drove this study: Increased interaction with the interventions was positively associated with increased social activity, more positive attitudes toward learning with the help of peers, higher performance on course programming assignments, and increased social connectedness. According to social learning theory, learning thrives when students are more active socially, are more positive about learning with the help of their peers and develop stronger connections to their peers.

However, it is important to keep in mind that the evidence produced in this study is *correlational*, not causal. We cannot conclude that the interventions themselves caused the increases in social behavior, more positive attitudes, and improved programming performance we observed. Rather, we can conclude that students who interacted with the interventions more frequently tended to be more active socially, tended to have more positive attitudes regarding peer learning, tended to be more socially connected, and tended to perform better on programming assignments. It could be the case that students who were *a priori* more socially active, more positive about peer learning, more socially connected, and higher academic achievers would tend to interact more frequently with the interventions. Clearly, the nature of the relationship between intervention interaction, social activity, attitudes, social connectedness, and programming performance remains an important open question for future research.

5 CONCLUSIONS

This article has motivated and presented the design of a set of software interventions embedded within a social IDE. Based on social learning theory, we posited that the interventions would lead to significant increases in students' social activity, which in turn would lead to improved learning outcomes, positive shifts in attitudes, and greater social connectedness. Based on an empirical evaluation, students varied considerably in terms of how much they interacted with the scaffolds—so much so that we were unable to identify causal links between intervention interaction and our dependent measures in an initial quasi-experiment. Given this, we have presented here a correlational analysis that provides evidence that intervention interaction is positively correlated with increased social activity, positive attitudes toward peer learning, greater social connectedness, and improved performance on course programming assignments.

5.1 Contributions

This research makes two key contributions to the literature on educational technologies for computing education:

1. *Design: A novel set of social interventions.* Based on social learning theory, we have introduced a set of novel software interventions (prompts and scaffolds) that can be embedded within an IDE and/or LMS to guide students toward educationally beneficial social interactions at strategic points in the learning process. The design of our interventions, together with our method for delivering the interventions through both an IDE and LMS, provides

a model for computing educators interested in building theoretically grounded online environments to foster more socially connected learning communities in their classrooms.

2. *Empirical: Study of social interventions.* We found that students varied considerably in the frequency of their interaction with the interventions, with some students ignoring them altogether and others interacting with them more frequently. Moreover, we presented evidence that increased interaction with the software interventions is positively correlated with increased social activity, positive attitudes toward peer learning, more closely coupled social networks, and improved performance in individual programming assignments. A deeper investigation of student interaction with the eight individual interventions developed in this work provides further insights into the extent to which students are inclined to interact with individual interventions designed to promote help-seeking, help-giving, topical contributions to online discussions, and self-reflection on one's programming process.

5.2 Limitations

This research has several notable limitations. We describe six of them below.

1. This study was able only to identify *correlations* between students' interaction with the interventions and increased social behaviors, positive attitudes, social connectedness, and academic performance; it was unable to assert that students' interaction with the interventions were *causally linked* to those variables.
2. While we found that students differed substantially with respect to their usage of the interventions, we were unable to identify factors that might predict students' level of usage of the interventions. Indeed, the slate of attitudinal variables measured in this study's pre-survey did not correlate with intervention usage. Future work should address this limitation by, for example, considering whether prior academic performance or other attitudinal and psychometric variables might correlate with intervention usage.
3. Our empirical studies tested eight different interventions all at once. This made it difficult to detect and understand the effects of the individual interventions. A better approach would have been to evaluate each intervention independently.
4. There are several online social environments into which social interventions might be integrated, including the Q&A interface supported by Stack Overflow [77], and the wiki-style knowledge repository supported by Piazza [56]. This research focused on a third possibility: the activity stream interface used in social media sites such as Facebook [19]. In so doing, it failed to consider the potential for interventions to enhance social interaction in other types of online social environments.
5. Our evaluation studies took place in face-to-face classes. Thus, students certainly interacted with each other outside of the online environment in which we collected data. Moreover, students certainly sought online help outside of our online environment (e.g., through internet searches). These outside interactions, which may have played a significant role in students' learning experiences, could not be captured through our online data collection process. This is clearly a significant limitation of this work that should be taken into consideration when interpreting our findings.
6. Our research was limited by the logistics of the CS1 courses studied. In an early empirical study, we studied a course whose instructor did not buy in to the use of the online social environment studied in this work. We believe this greatly diminished the use of the social environment by students in the course, as suggested by another research [48, 49]. To address this issue, we ran the study presented in this work in a course taught by the first author, who bought in to the online social environment and required its use as part

of the course. However, because the first author was privy to the design of the research while serving as a participant in the study, he may have biased the results. Clearly, a better approach would have been to have studied a course whose instructor was not involved in the design of the research.

5.3 Threats to Validity

This research is subject to several threats to validity. We elaborate below on four of the most serious of these.

1. *Statistical.* Our evaluation study was constrained by the number of students enrolled in the two relatively small CS1 courses included in the study ($n = 41$). As a result, the work is subject to a *low power* threat stemming from its sample size [75]. In addition, the study was vulnerable to an *unreliable implementation of intervention* threat since students may not have been exposed to the same interventions and course experiences. We aimed to mitigate these threats by requiring student usage of the online environment in which the interventions were delivered, and by implementing the course uniformly across offerings.
2. *Internal.* This study explored correlations between intervention usage and social interaction, attitudes, social connectedness, and learning outcomes. A threat to the validity of this approach is that factors apart from students' intervention interaction could also account for the correlations we observed—for example, prior academic ability. We attempted to mitigate this threat by considering relationships between pre-intervention attitudinal variables and students' level of interaction with the interventions. Future work could provide a broader picture of the associations identified in this work by including additional variables in such an analysis, including students' prior academic achievement, students' prior programming experience, students' prior relationships with students in the course, and other factors that could be related to their social interaction, social connectedness, attitudes, and academic performance. Indeed, we cannot rule out the possibility that factors not considered in this study may have stronger predictive power than student interaction with the interventions.
3. *Construct.* We evaluated students' social behavior solely through their participation in the online activity feed. Our reliance on a single measure of social behavior makes the research vulnerable to mono-operation bias. This bias was partially mitigated through our concurrent use of well-established attitudinal surveys, which provided a second measure of students' sociability.
4. *External.* The findings of this study are based on a sample of 41 students enrolled in two CS1 course offerings at our home institution. While the statistical methods used to support the findings were appropriate for this sample size, one must clearly exercise caution in any attempt to generalize the study's findings beyond the specific population studied. Future studies could address this threat by running comparative studies in a variety of courses at additional institutions.

5.4 Future Work

This work lays a foundation for future research into alternative ways to foster the kinds of social interaction and social learning communities that can promote positive learning processes, outcomes and attitudes in computing courses. One avenue for future research is to explore how to promote social interaction in alternative online social environments, including Q&A forums (e.g.,

[77]) and social wikis (e.g., [56]). The results of studies in alternative environments could be compared to identify ways in which the different environments differ with respect to their ability to promote desired behaviors, attitudes and outcomes.

Another avenue for future work is to refine the trigger system that determines when and which interventions to generate for a user. We focused here on a simple system that identified when counts of certain behaviors or time intervals reached threshold values. Future studies could explore more advanced algorithms that rely on, e.g., predictive models and states (Watwin score [83, 84], NPSM [15]).

We identified as a limitation of this study our simultaneous evaluation of a group of eight different interventions. Although we were able to note differences in the levels of interaction promoted by each intervention, we were unable to isolate the individual effectiveness of any one intervention; all interventions were tested as a group. To better understand the effectiveness of individual interventions, future work should take a more incremental approach in which interventions are tested individually before being combined. In addition, future work should model all variables considered in this study (engagement with the interactions, social behaviors, attitudes, social connectedness, and academic performance) collectively, to gain further insight into possible relationships among these variables.

Students demonstrated a reluctance to participate in the activity feed, ultimately compelling us to require a minimal level of participation. To help students overcome an initial reluctance to participate, future work would benefit from seeding initial discussions to jump-start student participation [43]. This is especially important for larger courses where students may be reluctant to make posts and replies if they do not see others making posts and replies.

Finally, to increase the chances that students will actively participate in online social environments, future work should develop training materials to help instructors understand system features and usage; to highlight the importance of having learners use the system and the impact such usage may have on learning outcomes and success; to provide guidelines for active instructor participation in the social feed; and ultimately to ensure that the instructor implements the pedagogy with high *implementation fidelity*. This is especially important considering the huge impact instructors' perceptions can have when they decide to adopt new tools and technology [48, 49]. Further, a recent education research report [38] suggests an important correlation between instructional format, teaching technique, instructor training, and larger treatment effect sizes, thus emphasizing the importance of instructor training and buy-in in future research.

ACKNOWLEDGMENTS

This work is based on the first author's doctoral dissertation, supervised by the second author at our home institution. We are grateful to the students who participated in our empirical studies.

REFERENCES

- [1] A. Aiken. 2020. A system for detecting software similarity.
- [2] H. Akoglu. 2018. User's guide to correlation coefficients. *Turkish Journal of Emergency Medicine* 18, 3 (2018), 91–93. DOI: <https://doi.org/10.1016/j.tjem.2018.08.001>
- [3] V. Aleven et al. 2003. Help seeking and help design in interactive learning environments. *Review of Educational Research* 73, 3 (2003), 277–320. DOI: <https://doi.org/10.3102/00346543073003277>
- [4] N. R. Aljohani and H. C. Davis. 2013. Learning analytics and formative assessment to provide immediate detailed feedback using a student centered mobile dashboard. *2013 Seventh International Conference on Next Generation Mobile Apps, Services and Technologies (NGMAST)* (2013), 262–267.
- [5] J. R. Anderson and B. Reiser. 1985. The LISP tutor. *Byte* 10, 4 (1985), 159–178.
- [6] R. A. Armstrong. 2014. When to use the Bbonferroni correction. *Ophthalmic and Physiological Optics* 34, 5 (2014), 502–508. DOI: <https://doi.org/10.1111/opo.12131>

- [7] A. W. Astin. 1999. Student involvement: A developmental theory for higher education. *Journal of College Student Development* 40, 5 (1999), 518–529.
- [8] A. Bandura. 1990. Perceived self-efficacy in the exercise of personal agency. *Applied Sport Psychology* 2, (1990), 128–163.
- [9] A. Bandura. 1997. *Self-efficacy: The Exercise of Control*. Worth Publishers.
- [10] J. Barr and A. Gunawardena. 2012. Classroom salon: A tool for social collaboration. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education (SIGCSE'12)*. 197–202.
- [11] J. Bennedsen and M. E. Caspersen. 2007. Failure rates in introductory programming. *ACM SIGCSE Bulletin* 39, 2 (2007), 32–36. DOI : <https://doi.org/10.1145/1272848.1272879>
- [12] K. Brennan. 2009. Scratch-Ed: an online community for scratch educators. *Proceedings of the 9th International Conference on Computer Supported Collaborative Learning* (Rhodes, Greece, 2009), 76–78.
- [13] C. J. Butz et al. 2006. A web-based Bayesian intelligent tutoring system for computer programming. *Web Intelli. and Agent Sys* 4, 1 (2006), 77–97.
- [14] P. J. Carrington et al. 2005. *Models and Methods in Social Network Analysis*. Cambridge University Press.
- [15] A. S. Carter et al. 2015. The normalized programming state model: Predicting student performance in computing courses based on programming behavior. In *Proceedings of the Eleventh Annual International Conference on International Computing Education Research*. ACM, 141–150.
- [16] A. S. Carter and C. D. Hundhausen. 2011. A review of studio-based learning in computer science. *Journal of Computing Sciences in Colleges* 27, 1 (Oct. 2011), 105–111.
- [17] A. S. Carter and C. D. Hundhausen. 2015. The design of a programming environment to support greater social awareness and participation in early computing courses. *J. Comput. Sci. Coll* 31, 1 (2015), 143–153.
- [18] A. S. Carter and C. D. Hundhausen. 2016. With a little help from my friends: An empirical study of the interplay of students' social activities, programming activities, and course success. *Proceedings of the 2016 ACM Conference on International Computing Education Research*. ACM. 201–209.
- [19] Facebook. 2016. <https://www.facebook.com/>. Accessed: 2016-03-27.
- [20] K. Falkner et al. 2014. Identifying computer science self-regulated learning strategies. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (New York, NY, USA, 2014), 291–296.
- [21] E. F. Gehringer. 2001. Electronic peer review and peer grading in computer science courses. *Proceedings of the 32nd SIGCSE Technical Symposium on Computer Science Education*. ACM Press. 139–143.
- [22] E. F. Gehringer et al. 2006. Expertiza: Students helping to write an ood text. *OOPSLA'06: Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications*. ACM Press. 901–906.
- [23] Gephi - The Open Graph Viz Platform: <https://gephi.org/>. Accessed: 2018-12-29.
- [24] M. Goldman et al. 2011. Collabode: Collaborative coding in the browser. *Proc. 4th Int. Workshop on Cooperative and Human Aspects of Software Engineering*. ACM. 65–68.
- [25] J. D. Gould and C. Lewis. 1985. Designing for usability: Key principles and what designers think. *Commun. ACM* 28, 3 (1985), 300–311. DOI : <https://doi.org/10.1145/3166.3170>
- [26] M. Guzdial. 1994. Software-realized scaffolding to facilitate programming for science learning. *Interactive Learning Environments* 4, 1 (1994), 1–44.
- [27] J. Hage et al. 2011. Plagiarism detection for Java: A tool comparison. *Computer Science Education Research Conference* (Heerlen, Netherlands, 2011), 33–46.
- [28] B. Hartmann et al. 2010. What would other programmers do: Suggesting solutions to error messages. *Proc. 28th Conference on Human Factors in Computing Systems*. ACM. 1019–1028.
- [29] C. D. Hundhausen et al. 2008. Exploring studio-based instructional models for computing education. *Proc. 39th SIGCSE Technical Symposium on Computer Science Education*. ACM Press. 392–396.
- [30] C. D. Hundhausen et al. 2010. The design of an online environment to support pedagogical code reviews. *Proceedings of the 41st ACM Technical Symposium on Computer Science Education*. ACM. 182–186.
- [31] V. Hyrynen et al. 2010. MyPeerReview: An online peer-reviewing system for programming courses. *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (New York, NY, USA, 2010), 94–99.
- [32] M. C. Jadud. 2006. Methods and tools for exploring novice compilation behaviour. *Proceedings of the Second International Workshop on Computing Education Research*. ACM. 73–84.
- [33] J. Jenkins et al. 2012. Perspectives on active learning and collaboration: JavaWIDE in the classroom. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, NC, USA, 2012), 185–190.
- [34] J. Johnson. 2010. *Designing with the Mind in Mind: Simple Guide to Understanding User Interface Design Rules*. Morgan Kaufmann Publishers Inc.
- [35] D. A. Kolb. 1984. *Experiential Learning: Experience as the Source of Learning and Development*. Prentice Hall.
- [36] M. Kölling et al. 2003. The bluej system and its pedagogy. *Journal of Computer Science Education* 13, 4 (2003), 249–268.

- [37] K. Kreijns et al. 2004. Determining sociability, social space, and social presence in (aA)synchronous collaborative groups. *CyberPsychology & Behavior* 7, 2 (2004), 155–172. DOI : <https://doi.org/10.1089/109493104323024429>
- [38] M. W. Lipsey et al. 2012. *Translating the Statistical Representation of the Effects of Education Interventions into More Readily Interpretable Forms*. Technical Report #NCSE 2013-3000. U.S. Department of Education. Institute of Education Sciences, National Center for Education Statistics.
- [39] W. Ma et al. 2014. Intelligent tutoring systems and learning outcomes: A meta-analytic survey. *Journal of Educational Psychology* 106, 2007 (2014), 901–918.
- [40] J. Maloney et al. 2010. The Scratch programming language and environment. *ACM Transactions on Computing Education* 10, 4 (2010), 1–15.
- [41] L. E. Margulieux et al. 2016. Employing subgoals in computer programming education. *Computer Science Education* 26, 1 (2016), 44–67. DOI : <https://doi.org/10.1080/08993408.2016.1144429>
- [42] C. McDowell et al. 2006. Pair programming improves student retention, confidence, and program quality. *Commun. ACM* 49, 8 (2006), 90–95. DOI : <https://doi.org/10.1145/1145287.1145293>
- [43] K. Miller et al. 2014. Improving online class forums by seeding discussions and managing section size. *Proceedings of the First ACM Conference on Learning @ Scale Conference* (New York, NY, USA, 2014), 173–174.
- [44] D. Mujumdar et al. 2011. Crowdsourcing suggestions to programming problems for dynamic web development languages. *CHI’11 Extended Abstracts on Human Factors in Computing Systems* (Vancouver, BC, Canada, 2011), 1525–1530.
- [45] J. C. Nesbit et al. 2014. How effective are intelligent tutoring systems in computer science education? *2014 IEEE 14th International Conference on Advanced Learning Technologies (ICALT)* (2014), 99–103.
- [46] R. S. Newman. 1998. Adaptive help seeking: A role of social interaction in self-regulated learning. *Strategic Help Seeking: Implications for Learning and Teaching*. Lawrence Erlbaum Associates Publishers. 13–37.
- [47] R. S. Newman. 1994. Adaptive help seeking: A strategy of self-regulated learning. *Self-regulation of Learning and Performance: Issues and Educational Applications*. Lawrence Erlbaum Associates, Inc. 283–301.
- [48] L. Ni et al. 2010. How do computing faculty adopt curriculum innovations? *The Story from Instructors*. (2010), 544–548.
- [49] L. Ni. 2009. What makes CS teachers change? Factors influencing CS teachers’ adoption of curriculum innovations. *ACM SIGCSE Bulletin* 41, 1 (2009), 544–548. DOI : <https://doi.org/10.1145/1508865.1509051>
- [50] J. Nielsen. 1993. Iterative user-interface design. *Computer* 26, 11 (1993), 32–41. DOI : <https://doi.org/10.1109/2.241424>
- [51] D. Norman. 2013. *The Design of Everyday Things: Revised and Expanded Edition*. Basic Books.
- [52] M. J. O’Grady. 2012. Practical problem-based learning in computing education. *Trans. Comput. Educ* 12, 3 (2012), 10:1–10:16. DOI : <https://doi.org/10.1145/2275597.2275599>
- [53] D. Olivares. 2019. *Exploring Social Interventions for Computer Programming: Leveraging Learning Theories to Affect Student Social and Programming Behavior*. Washington State University.
- [54] N. Parlante et al. 2016. Nifty assignments. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), 588–589.
- [55] T. V. Perneger. 1998. What’s wrong with Bonferroni adjustments. *BMJ*. 316, 7139 (1998), 1236–1238.
- [56] Piazza • Ask. Answer. Explore. Whenever. <http://www.piazza.com>. Accessed: 2019-11-21.
- [57] D. Pintrich et al. 1991. *A Manual for the Use of the Motivated Strategies for Learning Questionnaire*. Technical Report #NCRIPTAL-91-B-004. National Center for Research to Improve Postsecondary Teaching and Learning.
- [58] J. G. Politz et al. 2014. CaptainTeach: Multi-stage, in-flow peer review for programming assignments. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (New York, NY, USA, 2014), 267–272.
- [59] L. Porter et al. 2016. A multi-institutional study of peer instruction in introductory computing. *Proceedings of the 47th ACM Technical Symposium on Computing Science Education* (New York, NY, USA, 2016), 358–363.
- [60] L. Porter et al. 2011. Peer instruction: Do students really learn from peer discussion in computing? *Proceedings of the Seventh International Workshop on Computing Education Research*. ACM. 45–52.
- [61] Quasi-Experimental Design. 2006. <http://www.socialresearchmethods.net/kb/quasiexp.php>. Accessed: 2016-04-09.
- [62] A. D. Radermacher and G. S. Walia. 2011. Investigating the effective implementation of pair programming: An empirical investigation. *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (New York, NY, USA, 2011), 655–660.
- [63] V. Ramalingam and S. Weidenbeck. 1998. Development and validation of scores on a computer programming self-efficacy scale and group analyses of novice programmer self-efficacy. *J. Educational Computing Research*. 19, 4 (1998), 367–381.
- [64] K. Reily et al. 2009. Two peers are better than one: Aggregating peer reviews for computing assignments is surprisingly accurate. *Proceedings of the ACM 2009 International Conference on Supporting Group Work* (Sanibel Island, Florida, USA, 2009), 115–124.
- [65] I. Roll et al. 2011. Improving students’ help-seeking skills using metacognitive feedback in an intelligent tutoring system. *Learning and Instruction* 21, 2 (2011), 267–280. DOI : <https://doi.org/10.1016/j.learninstruc.2010.07.004>

- [66] G. Rößling et al. 2008. Enhancing learning management systems to better support computer science education. *SIGCSE Bull* 40, 4 (2008), 142–166. DOI: <https://doi.org/10.1145/1473195.1473239>.
- [67] M. B. Rossou et al. 2011. Orientation of undergraduates toward careers in the computer and information sciences: Gender, self-efficacy and social support. *Trans. Comput. Educ* 11, 3 (2011), 1–23.
- [68] J. B. Rotter. 1966. Generalized expectancies for internal versus external control of reinforcement. *Psychological Monographs: General and Applied* 80, 1 (1966), 1–28. DOI: <https://doi.org/10.1037/h0092976>
- [69] A. P. Rovai. 2002. Development of an instrument to measure classroom community. *Internet and Higher Education* 5, (2002), 197–211.
- [70] A. M. Ryan et al. 2001. Avoiding seeking help in the classroom: Who and why? *Educational Psychology Review* 13, 2 (2001), 93–114. DOI: <https://doi.org/10.1023/A:1009013420053>
- [71] A. M. Ryan et al. 2009. Do gender differences in help avoidance vary by ethnicity? An examination of African American and European American students during early adolescence. *Developmental Psychology* 45, 4 (2009), 1152–1163. DOI: <https://doi.org/10.1037/a0013916>
- [72] A. M. Ryan and P. R. Pintrich. 1997. “Should I ask for help?” The role of motivation and attitudes in adolescents’ help seeking in math class. *Journal of Educational Psychology* 89, 2 (1997), 329–341. DOI: <https://doi.org/10.1037/0022-0663.89.2.329>
- [73] S. Salinger et al. 2010. Saros: An eclipse plug-in for distributed party programming. *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering* (Cape Town, South Africa, 2010), 48–55.
- [74] E. Seymour and N. Hewitt. 1996. *Talking About Leaving: Why Undergraduates Leave the Sciences*. Westview Press.
- [75] W. R. Shadish et al. 2002. *Experimental and Quasi-Experimental Designs for Generalized Causal Inference*. Houghton Mifflin Company.
- [76] Simon et al. 2019. Pass rates in introductory programming and in other STEM disciplines. *Proceedings of the Working Group Reports on Innovation and Technology in Computer Science Education* (New York, NY, USA, 2019), 53–71.
- [77] Stack Overflow: 2016. <http://stackoverflow.com/>. Accessed: 2016-11-21.
- [78] B. E. Vaessen et al. 2014. University students’ achievement goals and help-seeking strategies in an intelligent tutoring system. *Computers & Education*. 72, (2014), 196–208. DOI: <https://doi.org/10.1016/j.compedu.2013.11.001>
- [79] K. Verbert et al. 2013. Learning analytics dashboard applications. *American Behavioral Scientist* 57, 10 (2013), 1500–1509. DOI: <https://doi.org/10.1177/0002764213479363>
- [80] K. Verbert et al. 2014. Learning dashboards: An overview and future research opportunities. *Personal Ubiquitous Comput* 18, 6 (2014), 1499–1514. DOI: <https://doi.org/10.1007/s00779-013-0751-2>
- [81] K. Verbert et al. 2013. Learning dashboards: An overview and future research opportunities. *Personal and Ubiquitous Computing*. (2013). DOI: <https://doi.org/10.1007/s00779-013-0751-2>
- [82] L. S. Vygotsky. 1978. *Mind in Society*. Harvard University Press.
- [83] C. Watson et al. 2014. No tests required: Comparing traditional and dynamic predictors of programming success. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*. ACM. 469–474.
- [84] C. Watson et al. 2013. Predicting performance in an introductory programming course by logging and analyzing student programming behavior. *Proceedings of the 2013 IEEE 13th International Conference on Advanced Learning Technologies* (2013), 319–323.
- [85] C. Watson and F. W. B. Li. 2014. Failure rates in introductory programming revisited. *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education* (New York, NY, USA, 2014), 39–44.
- [86] D. Wood et al. 1976. The role of tutoring in problem solving. *Journal of Child Psychology and Psychiatry* 17, 2 (1976), 89–100. DOI: <https://doi.org/10.1111/j.1469-7610.1976.tb00381.x>
- [87] H. Wood and D. Wood. 1999. Help seeking, learning and contingent tutoring. *Computers & Education* 33, 2 (1999), 153–169.

Received November 2019; revised June 2021; accepted June 2021