The Design of a Programming Environment to Support Greater Social Awareness and Participation in Early Computing Courses

Adam S. Carter and Christopher D. Hundhausen
Human-centered Environments for Learning and Programming (HELP) Lab
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164
+1 509-335-4590
cartera@wsu.edu, hundhaus@wsu.edu

ABSTRACT

Given the growing and widespread use of online social networking tools, coupled with social learning theory's emphasis on learning through social participation in a community, we believe there is good reason to pursue the development of educational programming environments that support increased social interaction among learners. To that end, we have been exploring the design of a "social programming environment" to support students as they work on individual programming assignments, which are common in early computing courses. We present the design of OSBIDE, an Integrated Development Environment (IDE) plug-in that supports many common features of social networking environments, including an activity feed and a social recommender system that identifies learners who have encountered similar programming issues. We describe the iterative refinement of OSBIDE through its multi-semester deployment in CS 1 and CS 2 courses. Preliminary results indicate that students are generally enthusiastic about the environment's ability to make programming more social. OSBIDE lays a strong foundation for future research into the relationship between learners' programming and social activities, and their learning outcomes. It also has the potential to provide instructors with powerful resources for identifying and assisting struggling students.

Categories and Subject Descriptors

K.3.1 [**Computer Users in Education**]: *Collaborative learning;* K.3.2 [**Computer and Information Science Education**]: *Computer science education, Curriculum.*

General Terms

Design, Experimentation, Human Factors.

Keywords

Social programming, social learning theory, social networking tools, activity feed, CS 1, CS 2

1. INTRODUCTION

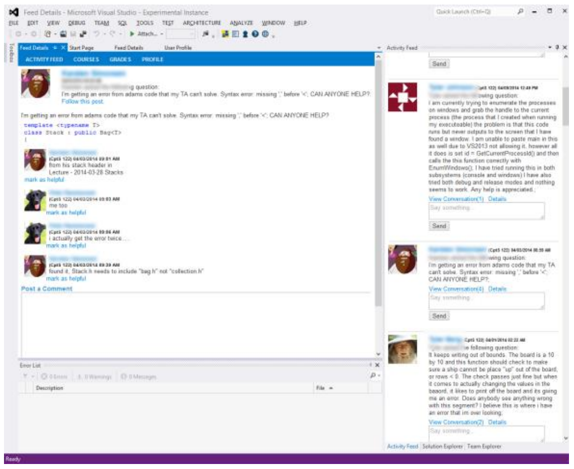
Especially in the younger generation, there is a growing trend toward the use of social media sites such as Facebook and Twitter. Indeed, in a recent survey of University of California students, students reported spending nearly as much time using social media as they did studying for class [1]. While it is easy to dismiss social media use as a waste of time, social learning theories such as Bandura's theory of self-efficacy [3] and Lave and Wenger's Situated Learning Theory [13] place great emphasis on social participation as a means to learning. This begs the following research question:

RQ1:	*Is it possible to leverage students' high use of social media for educational purposes?*

In previous research [11], we explored the use of Facebook as a discussion medium in early computing courses. While effective at facilitating conversations, we found that Facebook was often cumbersome to use as a means of discussing programming issues. We speculated that social media might be more beneficial if it could be more tightly integrated into the programming environment used by students to work on course programming assignments. This line of thinking led to the following research question:

RQ2:	*How can we best adapt features of social networking environments in order to build a more social programming environment?*

With a firm grounding in social theories of learning, we have been developing OSBIDE, a social media plugin for the Visual Studio IDE (see Figure 1). OSBIDE contains key features commonly found in social media: an activity feed, a social recommender system, profile pages, and user points and achievements. In addition, through the activity feed, it allows students to participate in conversations, pose questions, and search for solutions to problems within the context of programming. In this paper, we introduce and motivate the key design features of OSBIDE, briefly discuss the initial results of deploying the environment in early computing courses, and identify future research possibilities for the software.



[redacted] question:
I'm getting an error from adams code that my TA can't solve. Syntax error: missing ';' before '<'. CAN ANYONE HELP?
Follow this post

I'm getting an error from adams code that my TA can't solve. Syntax error: missing ';' before '<'. CAN ANYONE HELP?

```
template <typename T>
class Stack : public Bag<T>
{
```

[redacted]
(CptS 122) 04/03/2014 09:01 AM
from his stack header in
Lecture - 2014-03-28 Stacks
mark as helpful

[redacted]
(CptS 122) 04/03/2014 09:03 AM
me too
mark as helpful

[redacted]
(CptS 122) 04/03/2014 09:06 AM
i actually get the error twice....
mark as helpful

[redacted]
(CptS 122) 04/03/2014 09:39 AM
found it, Stack.h needs to include "bag.h" not "collection.h"
mark as helpful

Post a Comment

Send

[redacted] (CptS 122) 04/09/2014 12:49 PM
[redacted] following question:
I am currently trying to enumerate the processes on windows and grab the handle to the current process (the process that I created when running my executeable) the problem is that this code runs but never outputs to the screen that I have found a window. I am unable to paste main in this as well due to VS2013 not allowing it, however all it does is set id = GetCurrentProcessId() and then calls the this function correctly with EnumWindows(); I have tried running this in both subsystems (console and windows) I have also tried both debug and release modes and nothing seems to work. Any help is appreciated.;
View Conversation(1) Details
Say something...
Send

[redacted] (CptS 122) 04/03/2014 08:59 AM
[redacted] following question:
I'm getting an error from adams code that my TA can't solve. Syntax error: missing ';' before '<'. CAN ANYONE HELP?
View Conversation(4) Details
Say something...
Send

[redacted] (CptS 122) 04/01/2014 02:22 AM
[redacted] the following question:
It keeps writing out of bounds. The board is a 10 by 10 and this function should check to make sure a ship cannot be place "up" out of the board, or rows < 0. The check passes just fine but when it comes to actually changing the values in the baoord, it likes to print off the board and its giving me an error. Does anybody see anything wrong with this segment? I believe this is where i have an error that im over looking.
View Conversation(2) Details
Say something...

Figure 1: OSBIDE for Visual Studio

2. BACKGROUND AND RELATED WORK

Social learning theories served as the theoretical grounding for the construction of OSBIDE. According to Bandura's *self-efficacy theory* [3], one's perception of one's own abilities—so-called self-efficacy—is most influenced by *enactive experiences* (events directly experienced by an individual) and *vicarious experiences* (events witnessed by an individual). Positive self-efficacy has been strongly correlated with persistence within a discipline [2], including the discipline of computing [17]. OSBIDE provides opportunities for enactive experiences by allowing students to pose questions about their code to the class. It provides opportunities for vicarious experiences by allowing students to witness the successes and failures of their classmates' programming processes.

Similarly, Situated Learning Theory [13] holds that opportunities to participate, in increasingly central ways, in a community of practice is critical to learning. By integrating social media tools directly into students' problem solving environments, OSBIDE provides students with opportunities to participate peripherally in a community of learners by observing the programming activities and social interactions of others, and more centrally in the community by actually posing and answering questions that arise during the programming process.

While OSBIDE derives its greatest inspiration from contemporary social networking tools, it is related to a number of other software tools designed to facilitate collaborative learning and programming. One such family of tools aims to create online learning and teaching communities specifically focused on learning and teaching computer programming. For example, the Scratch novice programming environment [14] supports both an online learning community where learners share their programming creations, and an online teaching community where teachers share and discuss their teaching practices [5].

Environments such as Collabode [6, 7], JavaWIDE [12], and Saros [18] take a different approach: They enable programmers to collaborate on programming tasks over a networked environment. Common features include the ability to co-edit a document synchronously, see documents being edited by other users, and view document edits made by other users. Indeed, having students collaboratively program solutions, as embodied in the *pair programming* approach (see, e.g., [17]), has been shown to be an effective form of collaborative learning (see, e.g., [15]). However, these approaches prove incompatible with the kinds of programming scenarios targeted by this research: those in which students collaborate in the process of designing *their own* solutions to the same problem.

An alternative approach is to build specialized search tools to help novice programmers locate code written by others that is relevant to their current context. HelpMeOut [8] helps novice programmers fix programming errors by culling, from a central repository, relevant examples of how others fixed similar errors. In a similar vein, BluePrint [4] and CodeTrail [6] help novices locate and import relevant code examples by augmenting web searches with key contextual information from the IDE. In contrast, our research views programming not as a solo activity in which learners search for related examples when they get stuck, but rather as a *community* activity in which learners actively participate in the learning process by asking other community members (including experts) for help, and by offering help to others.

Within computing education, another line of related research is attempting to predict the performance of students based on programming behaviors within the IDE. A recent initiative within the BlueJ community has made the programming behaviors of hundreds of thousands of students from across the world available for data mining in the hopes of providing greater resources for computing education research [20]. Along these lines, Watson et al. [22] have reviewed and tested several approaches to using programming behaviors to predict academic success with great success. By adding *social behaviors* to the set of programming activities performed within an IDE, OSBIDE contributes a valuable new source of data that can be used to help understand programming behavior and predict academic success.

3. SYSTEM DESIGN

Throughout OSBIDE's design, we have continually sought to ground design decisions in the social learning theories introduced earlier [3, 13]. These theoretical frameworks place a heavy emphasis on allowing learners to observe the activities of others, and on promoting social interaction within a community of learners. We have attempted to embody these ideas with the following design goals:

1. Students should be able to observe others' activities and progress.
2. Students should be able to observe others' problem solving processes.
3. Students should be able to observe others' social interactions.
4. Students should be able to participate in a social learning community by
    a. Asking questions of others
    b. Answering other's questions

Table 1 explicitly connects these design goals with underlying theory and provides additional implications for design and pedagogy. The following vignettes expand upon our design goals and illustrate how these goals might manifest themselves in a social programming environment.

Type of Engagement	Theoretical Grounding	Implications for Design and Pedagogy
1. Observing others' activities and progress	Bandura: Observation of others' activities, progress, and process constitutes a *vicarious experience*, which impacts the self-efficacy of the observer. This impact is strengthened if the observer perceives the observed to be of similar ability. Lave & Wenger: Observation of others' activities, progress, and process constitutes a form of legitimate peripheral participation in a learning community. It forms a foundation for more central forms of participation.	• Enable learners to become aware of, and to observe, their peers' learning activities, especially those that are similar to their own [19]. • Present activities in a positive, encouraging light, using statements like "Don't give up! You're almost there!" [19]
2. Observing others' problem-solving processes		
3. Observing others' social interactions		
4. Asking questions of others	Bandura: Asking and questions are *enactive experiences*, the most powerful shaper of self efficacy. They can also increase the chances of task success, leading to more positive enactive experiences. Lave & Wenger: Asking and answering questions constitutes a more central form of community participation in a learning community.	• Lower the barriers to asking questions; make it easy to ask questions in the contexts in which they arise [16,21] • Provide incentives for answering questions to increase participation and increase chances that question-asking is perceived as helpful [9,19]
5. Answering others' questions		

Table 1: Forms of virtuous learner engagement enabled by OSBIDE, their theoretical benefits according to Bandura's and Lave and Wenger's Theories, and design and pedagogical strategies promoting them. The dotted line separates observational engagement from interactional engagement.

Vignette 1: Observing others' activities and progress

Jill is struggling to complete the "battleship" programming assignment. She doesn't understand how to reference a variable in one of her loops. Her code isn't compiling; she's getting a strange error. In the activity feed, Jill notices that other students recently got a similar error. This makes her feel slightly better.

Vignette 2: Observing others' problem solving processes

Jill decides to investigate a particular build error in further detail and clicks on the "See more…" link. She notices that one student, Jack, was able to compile his program just 5 minutes later. Tracing through the evolution of Jack's code, Jill sees that Jack fixed the error by adding an "&" before the variable. She immediately changes her code, and it successfully compiles.

Vignette 3: Observing others' social interactions

In the course of exploring events in the activity feed, Jill observes Jack and Jane interacting about code. She also notices that Jack, who recently had the same error she did, is asking the same kinds of questions she has wondered about. Others are helping him. Maybe if she asked, she could get the help she needed too?

Vignette 4: Asking questions of others

After executing her solution, Jill notices she's not getting the right answer. She highlights the code that she thinks could be the source of the problem and chooses "ask question" from the context menu. She types in "How can I get this loop to visit all coordinates?" and clicks "post." Her question, along with the code snippet she highlighted, is, posted to the activity feed, where it is immediately fielded by Dora, who just went through a similar struggle. Dora's response provides the insight Jill needs to fix the problem. She rates Dora's response as "helpful," which positively impacts Dora's community reputation rating.

Vignette 5: Answering others' questions

Later on, Jill notices that Dora is struggling with the code to test whether someone has won the game. Jill figured out that conditional test earlier, and decides to respond to Dora's query. Through multiple exchanges with Jill and Jim, a peer mentor who took the class last semester, Dora is able to understand how to formulate the conditional test properly. She rates Jill's comments as "helpful," which gives Jill her first reputation points.

We now turn to a discussion of the specific OSBIDE features that are designed to support these vignettes.

3.1 Activity Feed

Figure 2 presents our conceptualization of the activity feed. Analogous to Facebook's news feed, the activity feed is the focal point for social discourse within OSBIDE, and the starting point for investigating peer activity. Within the Visual Studio IDE, the activity feed exists in a window situated to the side of the main coding window. In this way, the feed acts as a peripheral awareness mechanism during a student's coding session.

The activity feed displays five types of items: generic feed posts, requests for help, marks indicating a response was found to be helpful, build errors, and runtime exceptions. The activity feed can be filtered based on a student's current objective. For example, Figure 2 depicts a feed that has been filtered to show only social activity. Note that the ability to filter the feed is directly integrated into the IDE: When a filter is applied (see Figure 3), the feed's contents are automatically updated.

As another example, Figure 4 depicts a feed that has been set to display only build errors. Note that OSBIDE anonymizes build errors. Originally, build errors and other programming mistakes that appeared in the activity feed were associated with the names of the students wo encountered them. However, we altered this design after receiving feedback that indicated that students were not always comfortable with their mistakes being publicly displayed to the rest of the class. Inspired by *self efficacy theory* [3], which holds that students need opportunities to evaluate themselves relative to others, we include an icon next to each build error that other learners have also encountered. Thus, students who encounter build errors can get the sense that they are not the only ones having problems; others are in the same boat.

3.2 Asking for Help

In a recent study of the use of activity streams in early computing courses, we found that activity streams are often used to pose coding questions [11]. To better support this practice, OSBIDE provides the ability for students to pose questions directly in the IDE by highlighting a block of code and then using a right-click context menu (see Figure 5). Using the context menu injects a student's question about the block of code directly into the activity feed to be seen by other students. This not only lowers the barrier to asking questions, but also ensures consistent code formatting and highlighting when viewing questions within OSBIDE. Furthermore, this allows students to pose questions in an *authentic* context—an important tenet of Situated Learning Theory.

3.3 Details View

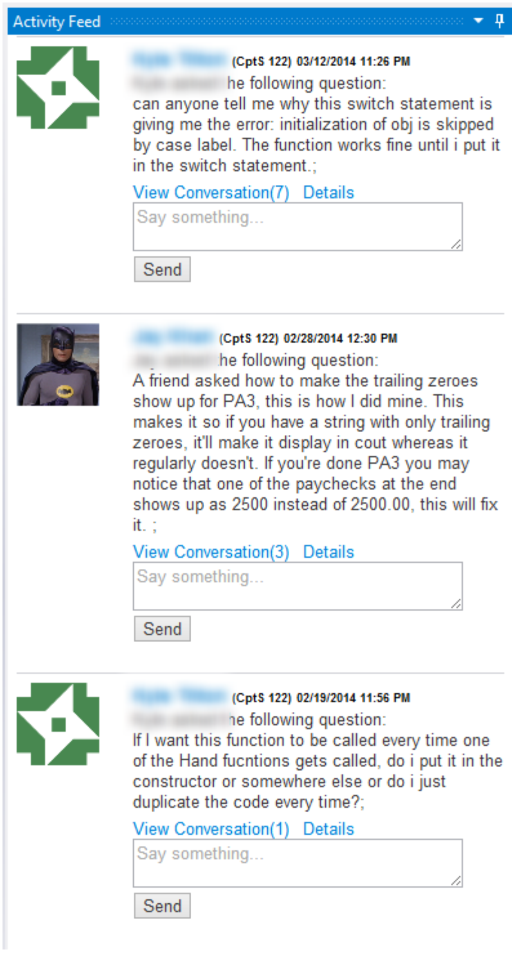
While the activity feed is meant to capture the class's stream of consciousness, the sidebar window in which the feed resides does not lend itself to displaying information in detail. Thus, OSBIDE provides a details view for each event type view (accessible through the "Details" link in Figure 2). Each details view contains additional information relevant to the specific event type. For example, the details view of an "Ask for Help" event (see Figure 5) might display the block of code selected by the user. For compile events, the details view displays the compilation error messages, the code around each error, and a "code diff" that displays how the compile error was addressed (see Figure 6). Similarly, a runtime exception event contains an entire stack trace, along with the values of relevant variables at the time of the runtime exception.

4. EVALUATION

We first piloted OSBIDE in the summer, 2013 offering (n = 18 students) and fall, 2013 offering (n = 223 students) of the CS 1 course at Washington State University. In these courses, students we required to install the OSBIDE plug-in as part of their normal course activities. These pilot deployments of OSBIDE brought to light several usability issues that contributed to the iterative refinement of the tool.

First, our original goal for the activity feed was to promote peripheral awareness of other students' programming activities, while as serving as the primary method of communication between students when out of class. We quickly found that the activity feed as designed could not simultaneously support both goals. Programming events tended to drown out social activity; for each social event, we discovered that there were roughly 20 programming events. This observation led to the development of improved filtering options, such as the ability to search for specific errors within the IDE, and the automatic promotion, within the activity feed, of social activity over automatically-generated programming events. In making these design changes, we transformed the activity feed from a peripheral awareness tool to a help-seeking tool.

Second, OSBIDE was originally an *optional* window within Visual Studio. This meant that students had to go through a series of menus in order to launch the tool. Results from a fall 2013 survey conducted in conjunction with the fall, 2013 deployment of

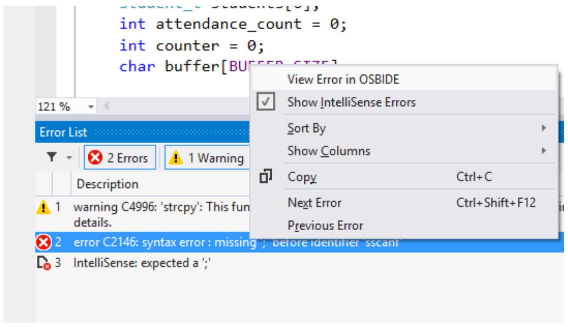


[redacted] (CptS 122) 03/12/2014 11:26 PM
[redacted] he following question:
can anyone tell me why this switch statement is giving me the error: initialization of obj is skipped by case label. The function works fine until i put it in the switch statement.;

[redacted] (CptS 122) 02/28/2014 12:30 PM
[redacted] he following question:
A friend asked how to make the trailing zeroes show up for PA3, this is how I did mine. This makes it so if you have a string with only trailing zeroes, it'll make it display in cout whereas it regularly doesn't. If you're done PA3 you may notice that one of the paychecks at the end shows up as 2500 instead of 2500.00, this will fix it. ;

[redacted] (CptS 122) 02/19/2014 11:56 PM
[redacted] he following question:
If I want this function to be called every time one of the Hand fucntions gets called, do i put it in the constructor or somewhere else or do i just duplicate the code every time?;

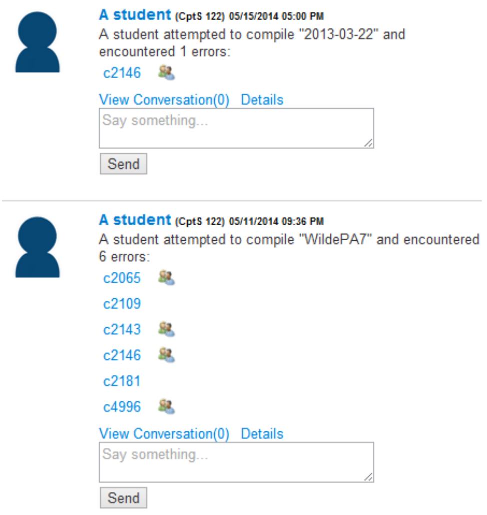
Figure 2: OSBIDE activity feed



```
int attendance_count = 0;
int counter = 0;
char buffer[BU
```

```
1 warning C4996: 'strcpy': This fun
  details.
2 error C2146: syntax error : missing
3 IntelliSense: expected a ';'
```

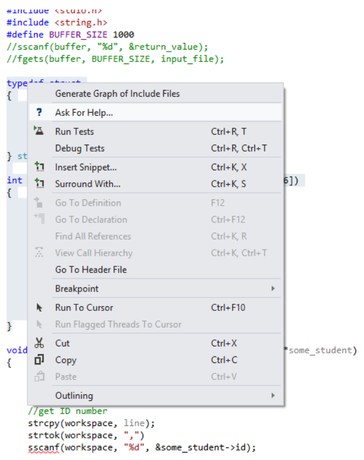
Figure 3: Using the activity feed to search for similar build errors



A student (CptS 122) 05/15/2014 05:00 PM
A student attempted to compile "2013-03-22" and encountered 1 errors:
c2146

A student (CptS 122) 05/11/2014 09:36 PM
A student attempted to compile "WildePA7" and encountered 6 errors:
c2065
c2109
c2143
c2146
c2181
c4996

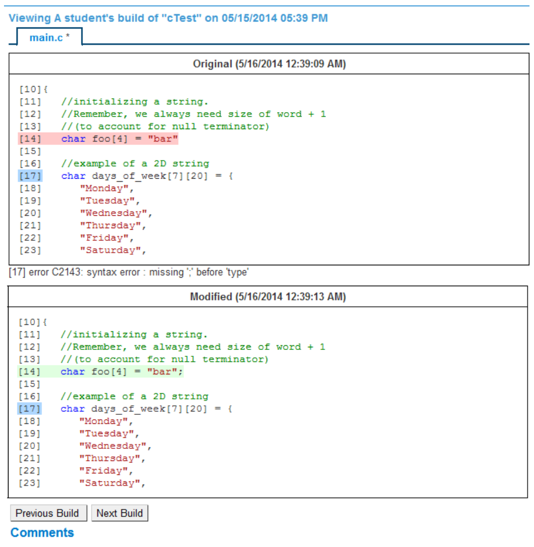
Figure 4: Activity feed filtered to build errors



```
#include <stdio.h>
#include <string.h>
#define BUFFER_SIZE 1000
//sscanf(buffer, "%d", &return_value);
//fgets(buffer, BUFFER_SIZE, input_file);

type
{


} st

int                                                    6])
{



}

void                                                   *some_student)
{

    //get ID number
    strcpy(workspace, line);
    strtok(workspace, ",")
    sscanf(workspace, "%d", &some_student->id);
```

Generate Graph of Include Files
Ask For Help...
Run Tests Ctrl+R, T
Debug Tests Ctrl+R, Ctrl+T
Insert Snippet... Ctrl+K, X
Surround With... Ctrl+K, S
Go To Definition F12
Go To Declaration Ctrl+F12
Find All References Ctrl+K, R
View Call Hierarchy Ctrl+K, Ctrl+T
Go To Header File
Breakpoint
Run To Cursor Ctrl+F10
Run Flagged Threads To Cursor
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Outlining

Figure 5: Asking for help within the IDE



Viewing A student's build of "cTest" on 05/15/2014 05:39 PM

main.c *

Original (5/16/2014 12:39:09 AM)

```
[10]{
[11]   //initializing a string.
[12]   //Remember, we always need size of word + 1
[13]   //(to account for null terminator)
[14]   char foo[4] = "bar"
[15]
[16]   //example of a 2D string
[17]   char days_of_week[7][20] = {
[18]      "Monday",
[19]      "Tuesday",
[20]      "Wednesday",
[21]      "Thursday",
[22]      "Friday",
[23]      "Saturday",
```

[17] error C2143: syntax error : missing ';' before 'type'

Modified (5/16/2014 12:39:13 AM)

```
[10]{
[11]   //initializing a string.
[12]   //Remember, we always need size of word + 1
[13]   //(to account for null terminator)
[14]   char foo[4] = "bar";
[15]
[16]   //example of a 2D string
[17]   char days_of_week[7][20] = {
[18]      "Monday",
[19]      "Tuesday",
[20]      "Wednesday",
[21]      "Thursday",
[22]      "Friday",
[23]      "Saturday",
```

Figure 6: Build error event details view

OSBIDE indicated that several students were unaware of the activity feed's existence. In order to make OSBIDE more visible, we decided to automatically launch the feed whenever a new instance of Visual Studio was launched.

Finally, OSBIDE was originally intended to exist solely within the IDE. However, a survey conducted in conjunction with the summer 2013 deployment of OSBIDE revealed that many students wished to access OSBIDE outside Visual Studio. In response, we made a version of OSBIDE accessible through the web.

Upon completing our original two pilot deployments, we introduced OSBIDE as a required component within the spring 2014 offering of the CS 2 course at Washington State University (n = 129 students). In a survey conducted in conjunction with this deployment of OSBIDE, student reactions were quite positive: 33% of students indicated that they checked the activity feed at least once a day, with 78% checking the feed at least twice a week. Moreover, 58% of students indicated that they were able to find answers through OSBIDE at least once per week, and 59% of students received at least one helpful suggestion per week. On the university's standardized course evaluation survey administered at the end of the semester, approximately 20% of students singled out OSBIDE as making a positive impact on their ability to learn course material.

In order to determine whether or not OSBIDE had an impact on learning outcomes, we examined each student's level of OSBIDE participation over the semester and related this to course deliverables (assignments, quizzes, exams, and final grades). A multivariate analysis of covariance revealed that regular OSBIDE usage was indeed a significant predictor of performance on all course deliverables, including students' final grade ($F >= 9.5$, $p < 0.01$). For a more in-depth analysis of these results, see [10].

5. CONCLUSION AND FUTURE WORK

In this paper, we have presented OSBIDE, an attempt to leverage social learning theories and the popularity of social networking in order to make individual programming assignments more social. Through the multi-semester design evolution described in this paper, we have iteratively refined the design of OSBIDE's social programming features, transforming its role from that of a social awareness mechanism to that of a help-seeking tool. Initial student feedback on OSBIDE has been positive, and our initial statistical analyses indicate a strong positive correlation between social activity in OSBIDE and course grades.

In future work, we plan to build on our design and analysis of OSBIDE in several ways. First, we would like to further investigate the programming behavior of students. Building upon the work of Watson et al. [22], we are in the process of designing and analyzing timeline visualizations of programming and social behavior. These visualizations categorize programming activity using a collection of distinct programming states (e.g. editing syntactically correct, semantically incorrect code; see Figure 7). We hypothesize that high-achieving students spend less time in "trouble" states and more time in "positive" states than lower-achieving students. In addition, we hypothesize that social activity is strongly correlated with transitions from "trouble" to "positive" states. Should our analyses yield worthwhile results, we hope to enable instructors to use the results to identify and work with struggling students earlier in semester.

Second, we wish to build upon our previous analyses of the relationship between the quantity and regularity of posts and academic success [10]. Taking these analyses one step further, we would like to examine the relationship between the *quality* and *content* of students' posts and their performance within the class. Furthermore, we would like to investigate how participation in the learning community relates to affective measures such as self-efficacy, sense of community, and peer learning.

Finally, we hope to perform an in-depth analysis of the community of practice supported by OSBIDE. We plan to identify distinct social roles based on patterns of social behavior (e.g. *lurkers*, *central contributors*, etc.), and to examine how these roles change over time. Of particular interest would be the investigation of the relationships between students' progression through the community of practice, their programming behaviors, and their overall academic success.

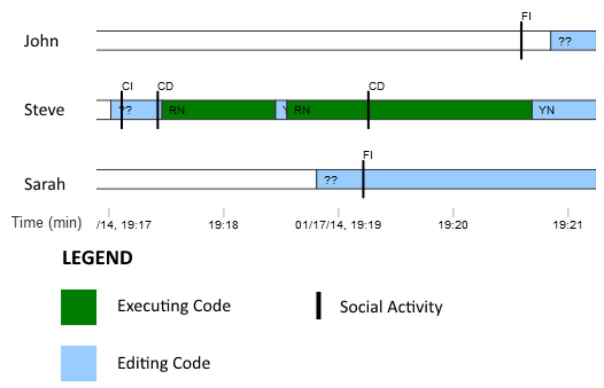


Figure 7: Timeline visualization of programming behaviors

6. ACKNOWLEDGMENTS

This project is funded by the National Science Foundation under grant no IIS-1321045.[19][16][21][9]

7. REFERENCES

[1] Arum, R. and Roska, J. 2011. *Academically adrift: limited learning on college campuses*. University of Chicago Press.

[2] Bandura, A. et al. 2001. Self-efficacy beliefs as shapers of children's aspirations and career trajectories. *Child Development*. 72, 1 (Feb. 2001), 187–206.

[3] Bandura, A. 1997. *Self-efficacy: the exercise of control*. Worth Publishers.

[4] Brandt, J. et al. 2010. Example-centric programming: integrating web search into the development environment. *Proc. 28th Conference on Human Factors in Computing Systems*. ACM. 513–522.

[5] Brennan, K. 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.

[6] Goldman, M. and Miller, R.C. 2009. Codetrail: connecting source code and web resources. *Journal of Visual Languages and Computing*. 20, 4 (2009), 223–235.

[7] Goldman, M. and Miller, R.C. 2011. Real-time collaborative coding in a web IDE. *Proceedings of the 24th annual ACM symposium on User interface software and technology* (Santa Barbara, CA, USA, 2011), 155–164.

[8] Hartmann, B. 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.

[9] Hattie, J. and Timperley, H. 2007. The power of feedback. *Review of Educational Research*. 77, (2007), 81–112.

[10] Hundhausen, C. et al. under review. Supporting programming assignments with activity streams: an empirical study. (under review).

[11] Hundhausen, C.D. and Carter, A.S. 2014. Facebook me about your code: an empirical study of the use of activity streams in early computing courses. *J. Comput. Sci. Coll.*. (2014).

[12] Jenkins, J. 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.

[13] Lave, J. and Wenger, E. 1991. *Situated Learning: Legitimate Peripheral Participation*. Cambridge University Press.

[14] Maloney, J. et al. 2010. The Scratch Programming Language and Environment. *Trans. Comput. Educ.*. 10, 4 (2010), 1–15.

[15] Radermacher, A. et al. 2012. Assigning student programming pairs based on their mental model of consistency: an initial investigation. *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (2012), 325–330.

[16] Roscoe, R.D. and Chi, M. 2007. Understanding tutor learning: Knowledge-building and knowldge-telling in peer tutors' explanations and questions. *Review of Educational Research*. 77, (2007), 534–574.

[17] Rosson, M.B. 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.

[18] Salinger, S. 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.

[19] Schunk, D.H. 2012. *Learning theories: An educational perspective*. Merrill Prentice Hall.

[20] Utting, I. et al. 2012. Web-scale data gathering with BlueJ. *Proceedings of the ninth annual international conference on International computing education research* (Auckland, New Zealand, 2012), 1–4.

[21] VanLehn, K. 2011. The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems. *Educational Psychologist*. 46, 4 (2011), 197–221.

[22] Watson, C. et al. 2014. No tests required: comparing traditional and dynmaic predictors of programming success. *Proceedings of the 45th ACM Technical Symposium on Computer Science Education* (2014), 469–474.

[23] Williams, L. and Kessler, R.R. 2001. Experimenting with industry's "pair-programming" model in the computer science classroom. *Computer Science Education*. 11, 1 (2001), 7–20.