Supporting Programming Assignments with Activity Streams: An Empirical Study

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

ABSTRACT

Social learning theory emphasizes the importance of providing learners with opportunities to observe their peers, and to participate actively in a community. Unfortunately, early computing courses tend to emphasize individual programming assignments, which discourage learners from observing and working with their peers. In order to explore the possibility that increased opportunities for social awareness and interaction while working on programming assignments might influence learning outcomes in early computing courses, we are studying the design and use of social networking-style activity streams in such courses. In an empirical study of the use of two types of activity streams in a CS 2 course—one that was part of a learning management system, and one integrated directly into students' programming environment—we found that students who used the integrated stream were twice as socially active; however, social participation in both environments was positively correlated with students' grades. Our results suggest that the use of activity streams as an adjunct to individual programming assignments can positively influence learning; computing instructors would do well to find ways to get their students to participate actively in activity streams during the programming process.

Categories and Subject Descriptors

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

General Terms

Design, Experimentation, Human Factors.

Keywords

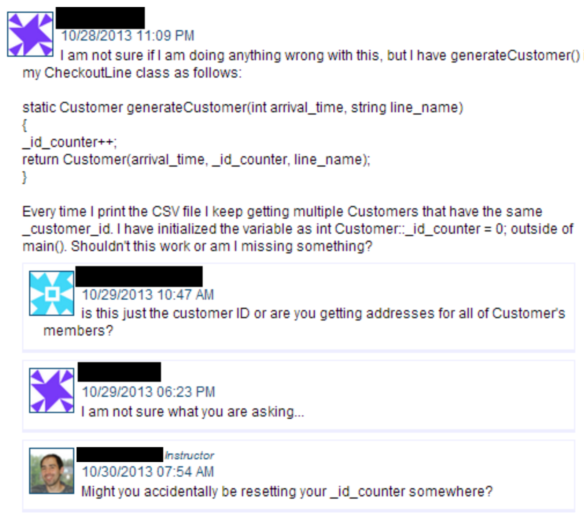
Social learning theory, programming assignments, Activity streams, social networking, CS 2

1. INTRODUCTION

An important component of early computing courses has traditionally been the *individual* programming assignment, which emphasizes individual problem solving and programming. In such assignments, students are typically discouraged or forbidden from observing and collaborating with their peers; they are asked to "go it alone," presumably to ensure that they learn the target skills and do not cheat.

There are at least two good reasons to call this traditional approach into question. First, the approach may not adequately prepare students for jobs in the computing profession, which, in addition to technical programming skills, are increasingly requiring so-called "soft" skills, including communication, teamwork, and collaboration (see, e.g., [3]). Second, the approach goes against contemporary social learning theory, which holds both that active participation in a community of peers and experts fosters learning [11], and that students have the best chance of sticking with the discipline if they are afforded opportunities to observe, and assess themselves relative to, their peers [1].

Given the above observations, computing educators would do well to consider ways of making the programming process less isolating and more social—especially in early computing courses, which typically suffer from high attrition rates [2]. Inspired by modern trends in social networking, we have been exploring the use of social networking-style *activity streams* (see Figure 1) to increase students' social awareness and engagement as they work on individual programming assignments. In social media



[redacted]
10/28/2013 11:09 PM
I am not sure if I am doing anything wrong with this, but I have generateCustomer() in my CheckoutLine class as follows:

```
static Customer generateCustomer(int arrival_time, string line_name)
{
_id_counter++;
return Customer(arrival_time, _id_counter, line_name);
}
```

Every time I print the CSV file I keep getting multiple Customers that have the same _customer_id. I have initialized the variable as int Customer::_id_counter = 0; outside of main(). Shouldn't this work or am I missing something?

[redacted]
10/29/2013 10:47 AM
is this just the customer ID or are you getting addresses for all of Customer's members?

[redacted]
10/29/2013 06:23 PM
I am not sure what you are asking...

[redacted] Instructor
10/30/2013 07:54 AM
Might you accidentally be resetting your _id_counter somewhere?

Figure 1. An Activity Stream in a Computing Course

environments such as Facebook, the activity stream enables people to participate in an asynchronous social community by sharing their life activities, as well as posing and answering questions. The idea extends naturally to learning communities of programmers: using an activity stream, learners are able to share their programming activities, and to ask and answer questions about programming, under the watchful eye of course teaching personnel, who can also contribute to the conversation as needed.

We suspect that activity streams are already commonly used by students during the programming process; however, no published research has investigated their instructional design and use in computing courses, or their impact on learning. Indeed, the use of activity streams as an adjunct to traditional individual programming assignments raises several research questions related to their use and educational effectiveness:

RQ1:	*To what extent do students appropriate an activity stream as they engage in computer programming assignments?*

RQ2:	*How is students' use of an activity stream related to their learning outcomes?*

RQ3:	*How does activity stream usage change over time, and what impact do changes have on performance?*

To explore these questions, this paper presents an empirical study of the use of activity streams in a CS 2 course at Washington State University. As an initial step toward exploring the design space of activity streams, we studied students' use of two alternative forms of activity streams during the programming process:

- A "traditional" activity stream that resides in a learning management system.
- An "integrated" activity stream that is embedded directly within the programming environment in which students complete course assignments.

Our results indicate that students were twice as active in the integrated activity stream as they were in the traditional activity stream. However, student participation within the activity stream positively correlated with course grades in both cases. As the first empirical study to consider a potential link between online social activity and programming outcomes, our study contributes to the empirical literature on the design of programming assignments, providing a theoretically-motivated and empirically-grounded basis for designing "social" programming assignments supported by an online activity stream.

2. BACKGROUND AND RELATED WORK

Social learning theory furnishes a compelling theoretical basis for efforts to make the learning of computer programming more social. Bandura's *self-efficacy theory* [1] posits that students develop a positive sense of their own programming abilities (so-called *self-efficacy*) by being able to observe the activities of their peers, and by being able to evaluate themselves relative to their peers. Not surprisingly, positive self-efficacy has been strongly correlated with persistence in the computing discipline [16]. Likewise, *situated learning theory* [11] holds that participation in a community of practice, which involves both the observation of others and actual participation in community activities, is essential to learning. Both of these social theories of learning suggest that learners may have difficulties making progress if they are forced to program in isolation from a broader community, as has traditionally been the case in computing education—especially in the individual programming assignments that are common in early computing courses.

Within the computing education research community, there have been several notable efforts to make computer programming more collaborative and social. One approach, with which this research has much in common, is to create online learning and teaching communities specifically focused on learning and teaching computer programming (see, e.g., [4, 12]). A second approach is to enable programmers to collaborate on programming tasks over a networked environment (e.g, [7, 10]). 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, the approach proves 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.

A third approach is to build specialized search tools to help novice programmers locate code written by others that is relevant to their current context (e.g., [6, 8]). 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.

Educational researchers have long been interested in understanding the relationship between learning processes and learning outcomes. In the field of Computer-Supported Collaborative learning (CSCL), researchers have studied online learning discourse, in an attempt to understand the kinds of discourse that might be associated with better learning outcomes (e.g., [5]). In a similar vein, computing educators have been interested in understanding the ways in which programming processes might correlate with programming and course outcomes (e.g., [17]). Within computing education, our research extends this work by exploring relationships between social behavior and learning outcomes.

3. EMPIRICAL STUDY

In order to explore the use and effectiveness of online activity streams in the process of completing programming assignments, we conducted a between subjects, quasi-experimental study with two treatments: *Traditional* and *Integrated*. These treatments corresponded to the fall 2013 and spring 2014 offerings of the CS 2 course at Washington State University. These two course offerings were identical in many important ways. Focusing on the C and C++ programming languages, both had three 50-minute lectures and one 170-minute lab period per week; three exams (two midterms and a final); and seven or eight individual programming assignments due at roughly two week intervals. Both courses were taught by the second author.

The two treatments were defined by the differing ways in which they implemented the use of activity streams as an adjunct to the individual programming assignments. In the *Traditional* treatment, students used an online activity stream that was part of the online learning management environment used for the course. In contrast, in the *Integrated* treatment, students used an activity stream directly integrated into the programming environment in which they completed the programming assignments.

A second difference between the two treatments was whether or not students were *required* to post to the activity stream. In the *Traditional* treatment, they were not, whereas in the *Integrated* treatment, they were required to make at least two posts, and to reply to at least two posts, per programming assignment.

Dependent measures included the number of posts and replies made (collected via transcripts of the activity streams), together with student grades—both the final course grade, and the grades for individual programming assignments and exams.

3.1 Participants

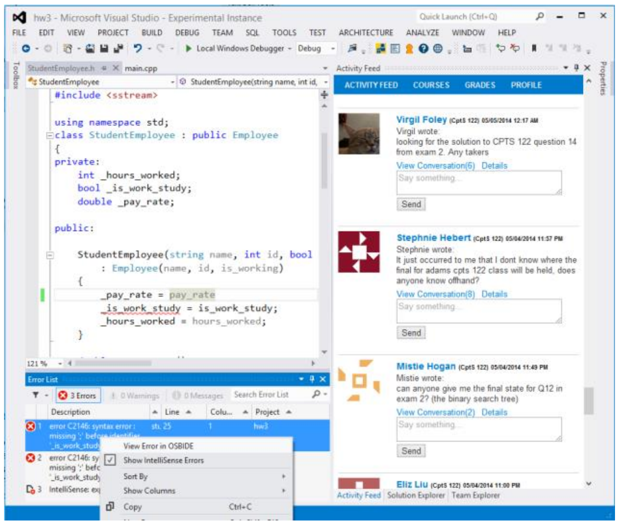
The fall 2013 course (the *Traditional* treatment) enrolled 150 students, of whom 135 completed the course with a grade. The spring 2014 course (the *Integrated* treatment) enrolled 140 students, of whom 129 completed the course with a grade. Students participated in the activity streams as part of their normal course activities. At the end of the semester, students were given the option of signing an informed consent form to release their activity stream posts and grades for research purposes. 102 students in the *Traditional* treatment, and 108 students in the *Integrated* treatment, consented to release their data.

3.2 Materials and Procedure

Students in the *Traditional* treatment used OSBLE [13], a learning management environment developed at Washington State University. In addition to hosting course materials and enabling the electronic submission of student assignments, OSBLE supported an online activity stream in which conversations could go one level deep (posts and replies; see Figure 1). Here, students were invited to engage in an online conversation about topics relevant to the course, although they were not required to participate in the conversation.

In contrast, students in the *Integrated* treatment used OSBIDE [14], a "social" plugin to Microsoft Visual Studio, the IDE in which students completed course assignments (see Figure 2). OSBIDE enabled students both to submit their assignments, and to participate in an online activity stream in which conversations could go one level deep.[1] Students in the *Integrated* treatment were *required* to make at least two posts per programming assignment, and to reply to at least two posts per programming assignment. Whereas students in the *Traditional* treatment completed eight programming assignments, students in the *Integrated* treatment completed only seven. Six of these were identical between the two treatments; a seventh was similar.

[1]OSBIDE supports additional social networking features, including the ability to view other students' programming activities (and their relation to one's own programming activities) in the activity stream, and the ability to highlight a block of code and ask a question about it in the activity stream; however, students' use of these features is not considered in the study presented here.



```
#include <sstream>

using namespace std;
class StudentEmployee : public Employee
{
private:
    int _hours_worked;
    bool _is_work_study;
    double _pay_rate;

public:

    StudentEmployee(string name, int id, bool
        : Employee(name, id, is_working)
    {
        _pay_rate = pay_rate
        _is_work_study = is_work_study;
        _hours_worked = hours_worked;
    }
```

Description	Line	Colu...	Project
1 error C2146: syntax error : missing ';' before identifier '_is_work_stud	stu. 25	1	hw3
2 error C2146: sy missing ';' befc '_is_work_study			
3 IntelliSense: ex			

Virgil Foley (CptS 122) 05/05/2014 12:17 AM
Virgil wrote:
looking for the solution to CPTS 122 question 14 from exam 2. Any takers
View Conversation(6) Details

Stephnie Hebert (CptS 122) 05/04/2014 11:57 PM
Stephnie wrote:
It just occurred to me that I dont know where the final for adams cpts 122 class will be held, does anyone know offhand?
View Conversation(8) Details

Mistie Hogan (CptS 122) 05/04/2014 11:49 PM
Mistie wrote:
can anyone give me the final state for Q12 in exam 2? (the binary search tree)
View Conversation(2) Details

Eliz Liu (CptS 122) 05/04/2014 11:00 PM

Figure 2. OSBIDE Plug-In to Microsoft Visual Studio

To perform the analyses presented in this paper, we extracted transcripts of the activity streams generated in each treatment. These transcripts, together with student grade data obtained from the course instructor, were exported to a database, enabling us to run queries to obtain the data to be analyzed.

3.3 Results and Discussion

We organize the presentation and discussion of our results around the three research questions we targeted for this study.

3.3.1 *To what extent is the activity stream used?*

Based on social learning theory [11] , which emphasizes the importance of *regular* participation within a learning community, we decided to consider two-week snapshots of student activity within each treatment—approximately the time interval between assignments. Our participation level metric, described in Table 1, ranges from 1 to 4, based on the number of posts and replies made by a student in a given two-week interval. To define the extremes of the metric, we defined level 1 as no participation at all, and level 4 as two or more posts and two or more replies—the requirement for full credit in the *Integrated* treatment. Levels 2 and 3 fall in between these two extremes.

Table 2 presents the mean participation level for each two-week interval, as well as the averages across the entire semester. As the table indicates, students in the *Integrated* treatment participated more actively in the activity stream. Indeed, a non-parametric Kruskal-Wallis ANOVA showed a statistically significant difference in level of participation between the two groups (H = 90.479, df = 1, p < 0.001), with students in the *Integrated* treatment maintaining a regular participation level that was nearly twice that of the *Traditional* treatment.

These results suggest that students will tend to appropriate an activity stream even if they are not required to do so, as was the case in the *Traditional* treatment. However, the combination of requiring students to participate in the activity stream and integrating the activity stream directly into an IDE appears to lead to a significant increase in students' participation level. Unfortunately, in this study, we were unable to disentangle the individual effect of making participation in the activity stream a requirement from the effect of integrating the activity stream into students' programming environment.

Table 1. Definition of Participation Level Metric

Level	Definition
1	No posts or replies
2	At least one post or reply, but no more than one of either
3	At least two posts and fewer than two replies, or at least two replies and fewer than two posts
4	Two or more posts and two or more replies

Table 2. Participation Level Averages by Two-Week Intervals (Standard Deviations in Parenthesis)

Weeks	Traditional (Fa 13)	Integrated (Sp 14)
1-2	1.20 (0.52)	1.68 (0.98)
3-4	1.46 (0.76)	2.66 (1.27)
5-6	1.31 (0.68)	2.73 (1.22)
7-8	1.39 (0.76)	2.63 (1.26)
9-10	1.24 (0.59)	2.16 (1.20)
11-12	1.41 (0.79)	2.30 (1.26)
13-14	1.26 (0.65)	2.59 (1.28)
15-16	1.39 (0.75)	2.31 (1.32)
Avg.	1.33 (0.70)	2.38 (0.89)

3.3.2 How is activity stream usage related to grades?

In order to determine whether or not regular participation was a reliable predictor of academic success, we considered each student's semester-long participation level average. In the case of the *Integrated* treatment, in which participation was mandatory, we were concerned that participation level might co-vary with prior computing grades: Since there was academic incentive to participate in the discussion, higher-achieving students might be more motivated to participate. Indeed, in the *Integrated* treatment, a correlational analysis between a student's level of participation and prior grade in CS 1 was found to be significant (r = .468, $p < 0.001$). However, in the *Traditional* treatment, this was not the case (r = .12, p = 0.26). Hence, we decided to retain prior CS 1 grade as a covariate in further analyses with the *Integrated* treatment (by using a MANCOVA), whereas we used a MANOVA for the Traditional treatment, while adding prior CS 1 grade as a separate independent variable for comparison purposes.

Tables 3 and 4 present the results of these analyses. As Table 3 shows, CS 1 grade was not a significant predictor of student performance in the *Traditional* treatment. In contrast, in the *Integrated* treatment, students' prior CS 1 grade was a significant predictor of students' exam scores and final grade. The partial eta squared (η^2) values indicate that the strength of the relationship between CS 1 grade and these two items was weak.

Table 4 tells a slightly different story: Participation level predicted student performance with respect to *all* graded items and the final grade in both treatments. Moreover, in all cases, the partial eta squared values indicate that participation level was a stronger predictor of student grades than was prior CS 1 grade, as it accounted for roughly twice the variance.

In order to further explore differences in students' grades vis-à-vis participation level, we partitioned students into quartiles based on their overall average participation level. Table 5 presents the mean final grade (percentage) by quartile. In both treatments, a between-subjects ANOVA detected a statistically-significant difference between the quartiles (T*raditional*: df = 3, F = 5.51, $p <$ 0.002; *Integrated*: df = 3, F = 11.62, $p < 0.001$). In the traditional treatment, a post-hoc Bonferoni test revealed a significant difference between the lowest and top-most quartile. A post-hoc Bonferoni test in the Integrated treatment revealed differences between the top two quartiles and the lowest quartile ($p < 0.01$) and the 2nd and 4th quartile ($p < 0.045$).

In order to determine the specific impact of participation quartile on student grades, we ran a regression analysis. In the case of the *Traditional* treatment, a change from one quartile to the next corresponded to a grade change of 6%. In the case of the *Integrated* treatment, a one-quartile change corresponded to a grade change of 4%. Thus, in both treatments, a one-quartile change equated to a difference of roughly one-half of a letter grade.

These results suggest that activity stream usage does, in fact, predict student grades in both treatments; students would do well to take a more active role in the activity stream. One possible explanation is that students who make and reply to posts are participating in valuable learning activities; active participation in the activity stream actually enhances learning. This explanation is favored by the social learning theories that form the foundation for this study. Indeed, according to Self-Efficacy Theory [1], while *vicarious* learning experiences in which students observe

Table 3. Extent to Which CS 1 Grade Predicted Course Grades

	Traditional (Fa13)			Integrated (Sp14)		
Graded Item	**F**	**Sig.**	η^2	**F**	**Sig.**	η^2
Assignments	0.26	0.61	0.00	1.39	0.24	0.01
Quizzes	1.86	0.18	0.02	3.81	0.54	0.04
Exams	0.68	0.41	0.01	9.88	**0.01**	0.09
Final Grade	0.04	0.95	0.00	6.67	**0.01**	0.07

Table 4. Extent to Which Participation Level (see Table 1) Predicted Course Grades

	Traditional (Fa 13)			Integrated (Sp 14)		
Graded Item	**F**	**Sig.**	η^2	**F**	**Sig.**	η^2
Assignments	6.74	**0.02**	0.06	10.5	**0.01**	0.10
Quizzes	6.08	**0.02**	0.06	14.9	**0.01**	0.13
Exams	6.33	**0.02**	0.06	9.50	**0.01**	0.09
Final Grade	13.2	**0.01**	0.17	23.2	**0.01**	0.20

Table 5. Final Grades (%) by Participation Level Quartile

	Traditional (Fa 13)			Integrated (Sp 14)		
Quartile	**N**	**M**	**SD**	**N**	**M**	**SD**
1	22	66.0	17.6	20	76.9	13.3
2	9	75.5	19.9	29	84.8	8.1
3	10	75.6	21.5	24	86.2	5.9
4	25	85.3	10.3	36	90.5	5.9

others without actively participating are valuable, *enactive* learning experiences in which students actively participate are absolutely essential.

3.3.3 How does activity stream usage change over time, and what impact do changes have on performance?

Figures 3 and 4 present, for each treatment, state transition networks depicting the probability that students will transition, from one two-week interval to the next, to each of the four participation levels defined in Table 1. In these networks, edges are labeled with their associated probabilities; thicker and redder edges indicate higher-probability transitions. As can be seen, the most likely transitions are self-transitions to the same level of participation at the two extremes (level 1 and level 4). In other words, high-participating students tended to continue to participate actively, while low-participating students tended to continue not participating.

To explore whether changes in participation levels over time influenced student grades, we first rank-ordered participants based on their participation level by time period; participants whose participation level increased over time were ranked higher. We then performed a multiple regression analysis, with final examination as the dependent variable and participation level quartile and participation ranking as independent variables.

With an adjusted R^2 value of .15, the regression model explained 15% of variance in student grades: 15% of the variability in grades was predicted by participation rank and level. In fact, participation rank was found to be a significant factor in the model in both treatments (p = .004): Students who increased their participation over time tended to be associated with higher grades.

This result suggests that students may learn better and obtain higher grades if they engage in *increasingly* active participation in the activity stream. This result is in line with *situated learning theory* [10]—especially the notion that increasingly central participation in a community of practice (which in our study equated to increasingly active participation in the activity stream) constitutes evidence of learning. Indeed, it appears that as the semester progressed, higher-achieving participants in our study were increasingly involved in the practices of the community and were more willing to participate.

4. CONCLUSION

In this paper, we have presented a comparative study of two alternative versions of activity streams in an early computing course. Our study sought to explore not only the extent to which students might use the alternative activity streams, but also the relationship between such usage and course performance. We found that students used the integrated activity stream significantly more than the traditional one. However, such increased usage did not appear to lead to any advantages in terms of course performance: in *both treatments*, participation level, as well as an increase in participation level, was positively correlated with course grades—a result that can be explained by social learning theory.

4.1 Limitations

It is important to point out three limitations of our study. First, it considered only the *quantity* of participation in the activity stream, without considering *quality*. Clearly, a more sophisticated analysis

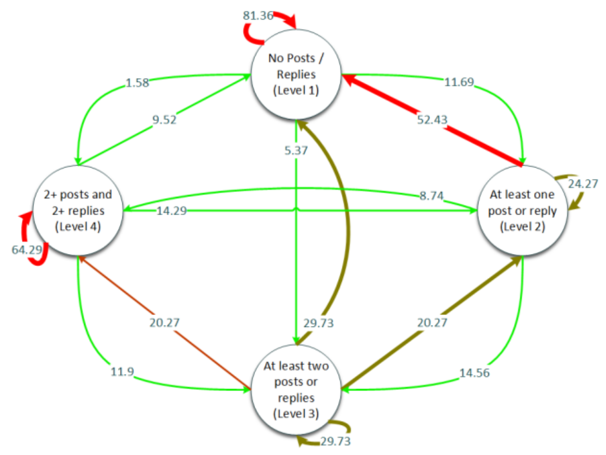

81.36
No Posts / Replies (Level 1)
1.58
11.69
9.52
52.43
5.37
2+ posts and 2+ replies (Level 4)
14.29
8.74
At least one post or reply (Level 2)
24.27
64.29
20.27
29.73
20.27
11.9
14.56
At least two posts or replies (Level 3)
29.73

Figure 3. Probabilistic State-Transition Diagram for Traditional Treatment

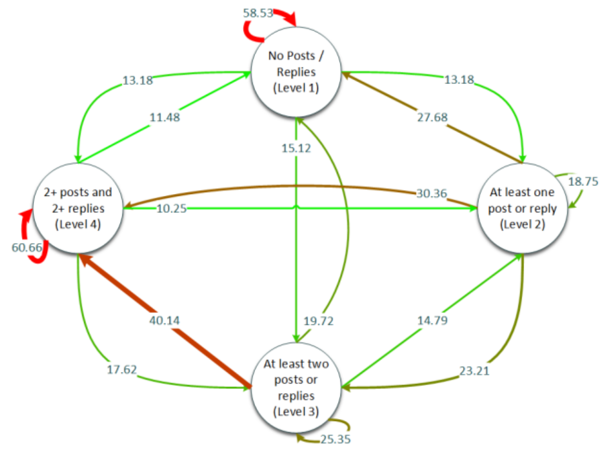

58.53
No Posts / Replies (Level 1)
13.18
13.18
11.48
27.68
15.12
2+ posts and 2+ replies (Level 4)
10.25
30.36
At least one post or reply (Level 2)
18.75
60.66
40.14
19.72
14.79
17.62
23.21
At least two posts or replies (Level 3)
25.35

Figure 4. Probabilistic State-Transition Diagram for Integrated Treatment

would have attempted to code students' activity stream posts based on their content, in order to explore more deeply relationships between the quality and focus of students' posts, and their course performance.

Second, this study considered two different CS 2 courses of moderate to large size at Washington State University taught by the second author of this paper. The fact that the second author—a stakeholder in the particular technology used in the *Integrated* treatment—was involved as a participant in these studies may have influenced his participation. Moreover, although we believe they are representative of typical CS 2 courses at many universities, caution must be exercised in any attempt to generalize this study's results beyond the two CS 2 courses considered in the study.

Third, because students in the *Integrated* treatment were required to post, whereas students in the *Traditional* treatment were not, this study was unable to disentangle the influence of posting as a requirement from the influence of integrating the activity feed into the IDE. Clearly, disentangling these two issues is an important topic for future study.

4.2 Implications for Computing Education

Our results have important implications for the instructional design of individual programming assignments in computing education. Integrating an online activity stream into students' out-of-class programming assignments appears to be a best practice that can yield positive benefits for student course performance. As this study has shown, and as social learning theory predicts, students who participate actively in an activity stream will tend to do better in the course. Instructors who can find ways to get their students to participate more actively in an activity stream can make a real difference in their students' course success.

At the same time, our study results have implications for the design of the activity stream used for course assignments. While an activity stream integrated directly into the IDE used by students will tend to promote more active student participation than a traditional activity stream not integrated into an IDE, such increased participation may not be associated with improved student performance. According to our results, instructors may do just as well to enlist a traditional activity stream supported by a learning management environment or commercial social networking tool.

4.3 Future Work

This preliminary research into the relationship between students' online social activity and their programming success raises intriguing possibilities for future research. In future work, we are interested in pursuing the following two research directions:

Activity stream post quality. As mentioned above, a key limitation of our study was that it focused squarely on the quantity of activity stream posts, without considering what was actually written in the posts. In a preliminary study conducted in a CS ½ and CS 1 course, we have performed a content analysis of activity streams [9]; however, that study made no attempt to correlate post content and quality with student grades. In ongoing work, we are performing a content analysis of the activity streams of the two CS 2 courses considered in this study, so that we can gain further insights into what students discuss in an activity stream and how the content and quality of their discussions relate to course performance.

Integrating students' programming activities into an activity stream. The OSBIDE social plug-in used in the *Integrated* treatment logs all of students' programming activities, including their edits, build errors, debugging events, and run-time exceptions. How can these data about programming activities be integrated into an activity stream, and what educational benefits might the inclusion of such data have? In ongoing work, we are exploring this design space, including the use of a social recommendation system to make students aware of other students who are encountering similar programming issues. The goal is to develop a new breed of *social* programming environments that leverage social learning theory to make introductory computing less isolating and more community-based.

5. ACKNOWLEDGMENTS

This project is funded by the National Science Foundation under grant no IIS-1321045.

6. REFERENCES

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

[2] Beaugouef, T. and Mason, J. 2005. Why the high attrition rate for computer science students: Some thoughts and observations. *ACM SIGCSE Bulletin*. 37, 2 (2005), 103–106.

[3] Begel, A. and Simon, B. 2008. Novice software developers, all over again. *Proceeding of the Fourth International Workshop on Computing Education Research*. ACM. 3–14.

[4] 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.

[5] Dennen, V.P. 2008. Looking for evidence of learning: Assessment and analysis methods for online discourse. *Computers in Human Behavior*. 24, 2 (2008), 205 – 219.

[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] 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).

[10] 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.

[11] Lave, J. and Wenger, E. 1991. *Situated learning: Legitimate peripheral participation*. Cambridge Univ Pr.

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

[13] Online Studio-Based Learning Environment: 2012. *https://osble.org/*. Accessed: 2012-01-14.

[14] OSBIDE - Home: *http://osbide.codeplex.com/*. Accessed: 2012-10-17.

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

[16] 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.

[17] 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.

[18] 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.