Exploring the Effects of Social Programming Environments on Novice Programmers

Adam Carter

School of Electrical Engineering and Computer Science

Washington State University
# Table of Contents

1. **Introduction** .................................................................................................................. 4  
   1.1 What Makes Learning to Program Difficult? ................................................................. 4  
   1.2 Why do Students Leave Computing Majors? ................................................................. 6  
   1.3 Thesis ............................................................................................................................. 7  
   1.4 Motivating Theory ......................................................................................................... 7  
   1.5 Research Questions ....................................................................................................... 9  
   1.6 Proposal Overview ........................................................................................................ 9  

2. **The Design Space of Social Programming Environments** ........................................... 10  
   2.1 Awareness Mechanism ................................................................................................. 10  
   2.2 Communication Mechanism ....................................................................................... 16  
   2.3 Pedagogy ..................................................................................................................... 17  

3. **Research Plan** .............................................................................................................. 19  
   3.1 Study Design ............................................................................................................ 19  
   3.2 Independent Variables ............................................................................................... 20  
   3.3 Dependent Variables ................................................................................................. 21  
   3.4 Pilot Study: Data Gathering Tool .............................................................................. 23  
   3.5 Pilot Study: SPE .......................................................................................................... 24  

4. **SPE Design** .................................................................................................................. 25  
   4.1 Logging Tool Design and Construction .................................................................... 25  
   4.2 Social Programming Component ............................................................................ 27  

5. **Related Literature** ...................................................................................................... 33  
   5.1 Theories of Learning ................................................................................................. 33  
   5.2 Learner Communities in Computing Courses ........................................................... 35  
   5.3 Novice Programming Environments ......................................................................... 36  
   5.4 Social Programming Aids ......................................................................................... 36  
   5.5 Cheating .................................................................................................................... 37  
   5.6 Privacy ....................................................................................................................... 38  
   5.7 Summary .................................................................................................................... 38  

6. **Appendix A: Online Survey** ....................................................................................... 39  
   6.1 General Demographic Data ....................................................................................... 39  
   6.2 Adapted C++ Self-Efficacy Survey .......................................................................... 39  
   6.3 Classroom Community Scale ................................................................................... 40  
   6.4 Motivated Strategies for Learning Questionnaire ..................................................... 40  

7. **Appendix B: Informed Consent** .................................................................................. 42  

8. **References** .................................................................................................................. 44
**List of Figures**

Figure 1: Design Space of Social Programming ................................................................. 11
Figure 2: Facebook Activity Stream .................................................................................. 12
Figure 3: User-Centered View ......................................................................................... 15
Figure 4: User Profile Page ............................................................................................ 15
Figure 5: Timeline Visualization ...................................................................................... 15
Figure 6: Code Annotation Example ............................................................................... 17
Figure 7: Stack Overflow Point System ............................................................................ 22
Figure 9: Event Logging Client Architecture ..................................................................... 26
Figure 10: Event Logging Server Architecture ................................................................. 26
Figure 11: Activity Stream within the Visual Studio IDE .................................................. 27
Figure 12: Components of the Activity Stream ............................................................... 29
Figure 13: Activity Post Details View ............................................................................... 30
Figure 14: User Profile View .......................................................................................... 30
Figure 15: Adding / Removing Users from Group ............................................................. 30
Figure 16: Asking for Help ............................................................................................... 31
Figure 17: Chat Interface ............................................................................................... 32

**List of Tables**

Table 1: Proposed Research Plan .................................................................................... 19
Table 2: Dependent and Independent Variables ............................................................. 23
Table 3: Pilot Data Averages .......................................................................................... 24
Table 4: Pilot Data Totals ............................................................................................... 24
1 Introduction
Learning to program is a difficult task that can be challenging to even the most intelligent students. When students fail to grasp computing concepts, it is easy to conclude that the task of learning computing concepts was simply too great a challenge. If retention rates in computing were near the national average of 78% (National Center for Education Statistics, U.S. Department of Education, 2011), it might be forgivable to look past the occasional computing dropout. Unfortunately, retention within the discipline of computing is much lower. According to recent statistics, only 46% of students who entered baccalaureate granting institutions with the intention to earn a degree in computing actually graduated within six years (National Center for Education Statistics, U.S. Department of Education, 2011). Another study found that in a comparison between science, technology, engineering, and mathematics (STEM) majors, students enrolling in computing majors were the least likely to complete their degree (Xianglei & Weko, 2009). In fact, computing has one of the lowest baccalaureate retention rates among all majors (National Center for Education Statistics, U.S. Department of Education, 2011). While we cannot expect to expect to completely solve the retention issue, it is realistic to expect the rates to rise to the levels of other fields of study. Given this state of affairs, I pose the following questions:

1. What makes learning to program difficult?
2. Why do students leave computing majors?

Investigating these questions will allow us to better understand the current academic climate and provide insights into future research directions.

1.1 What Makes Learning to Program Difficult?
When asked about students who struggle, it is easy to point to a ”geek gene” that, when present, enables success, and, when absent, guarantees failure (Lister, 2010; Seymour & Hewitt, 1996; Tinto, 1993). A strong counterargument to this claim is the work done by Seymour and Hewitt (1996), who interviewed undergraduate STEM majors in two categories: those who left STEM majors (“switchers”), and those who remained in STEM majors. Their multi-year, multi-institution study (n = 335) found that the majority of students who left STEM majors were just as academically capable as their non-leaving counterparts. If ability is not the cause of computing’s notoriously high attrition rate, what else might it be? Others have asked this question (Guzdial, 2004), and generally speaking, their answers can be placed into one of three categories, as described below.

1.1.1 Programming Environments / Languages
It does not take a great leap of imagination to conclude that industrial development environments such as Microsoft Visual Studio or Eclipse might not be ideally suited for teaching programming fundamentals (see, e.g. Storey et al., 2003). Industrial programming languages such as C or Java pose a similar dilemma. Research in this area generally focuses on trying to make the programming environment, language, or both more approachable for novices. Popular approaches include the pairing of abstract concepts with more concrete visual representations (see, e.g. Hundhausen, Farley, & Brown, 2009), reducing the number of programming commands (Guzdial, 2004), or preventing the user from writing incorrect syntax (Resnick et al., 2009). Several novice-oriented systems are described and categorized by Kelleher and Pausch (2005). Assessments of these systems vary greatly and usually include some form of subjective assessment from students and/or instructors along with examinations of students’ academic performance and/or retention. While these systems appear to promote achievement among low performing students, the lack of detailed, rigorous investigations prevent broader claims from being made (P. Gross & Powers, 2005).

1.1.2 Course Content
Rather than focus on a particular language or development environment, some researchers instead focus on how best to structure or present introductory course content. If this line of research were asked the
question of what makes learning to program difficult, it would likely respond, "the difficult course content." Language selection (Boszormenyi, 1998), the ordering of content (Pears et al., 2007), and programming model (e.g. objects first) (Ehler & Schulte, 2009) have all been presented as ways to improve learning or motivation. Another popular approach has been to structure curriculum content around a particular construct such as game development (see, e.g. Bayliss, 2009), or robot construction (see, e.g. van Delden & Wei, 2008). Perhaps even more so than the results on programming environments and languages, papers that investigate changing course content tend to be mixed with some studies reporting positive results (see, e.g. Bayliss, 2009; van Delden & Wei, 2008) while others report less of an effect (see, e.g. Ehler & Schulte, 2009; Simon et al., 2009). Regardless of the effectiveness of such treatments, restructuring course content tends to lack generalizability, as it heavily depends on the course instructor and the materials available. For example, language selection has a large impact on the types of games that can be developed or the robots that can be used. Furthermore, these implementations are not "one size fits all." While game development is known to be quite popular with male students, this is less true for female students (Gurer, 2002). Therefore, it makes sense to want to look beyond individual content adjustments by examining the larger pedagogical issues.

1.1.3 Pedagogies

Researchers interested in pedagogical research might argue that it's not what we teach, but how we teach it, that makes programming difficult to learn. To this end, pedagogical researchers often investigate augmenting or replacing the traditional lecture format with pedagogies of engagement (Smith, Sheppard, Johnson, & Johnson, 2005). Whereas lecture assumes that learning is acquired passively, pedagogies of engagement assume that learning is acquired when the learner actively engages with the material. Pedagogies of engagement vary along the dimensions of formality, scope, and collaboration. For example, Active Learning is an informal pedagogy of engagement in which students form ad-hoc groups during class to discuss lecture topics (Smith, Douglas, & Cox, 2009). At the other end of the formality and scope spectrum is Problem-Based Learning (PBL) where students work in cooperative groups for extended periods of time. Both active learning and PBL also vary on the amount of collaboration required. While all levels of collaboration have theoretical merit, the recent trend has been towards more socially (i.e. group) oriented pedagogies (see, e.g. Hundhausen, Agrawal, & Agarwal, In press; Smith et al., 2005).

While there are some dissenting opinions on the effectiveness of socially-oriented pedagogies, and pedagogies of engagement in general (see, e.g. Kirschner, Sweller, & Clark, 2006), most agree that socially oriented pedagogies are more effective than passive learning models (National Research Council, 2000; Smith et al., 2005). For example, a large meta-analysis on problem based learning (PBL) within the field of medicine found that a PBL-based curriculum promotes better interpersonal skills while also improving the perceived quality of education and lessening dropout rates (Schmidt, Van der Molen, & Te Winkel, 2009). Similarly, in a study of the use of algorithm visualization in computer science education, Hundhausen et al. (2002) found that student engagement was the main factor in determining pedagogical effectiveness. However, in spite of these mostly positive results, adoption of socially-oriented pedagogies within computing education remains limited. I discuss possible reasons for this below.

1.1.3.1 Logistical Issues

Several logistical factors may inhibit the adoption of socially-oriented pedagogies by computing majors. As previously stated, these pedagogies encourage a high level of activity with the course material and other students. However, the majority of classes are taught in lecture halls which, as their name implies, are intended to focus attention towards the front of the class and generally make active engagement difficult. Similarly, students need areas outside of class that are conducive to active engagement, which is often not the case. In many cases, lab rooms are organized much like lecture halls (Patitsas, 2012), thereby making collaboration and communication feasible only with one's immediate neighbors. Nevertheless, space related logistical issues are not insurmountable. Indeed, entire buildings and
departments have been restructured to accommodate active pedagogies (see, e.g. Lynch, Carbone, Arnott, & Jamieson, 2002). However, adopting such an approach in this manner is far beyond the scope of most universities.

Another logistical issue comes by way of the chosen design medium. Unlike other disciplines that start with analog representations (see Schon, 1983), most computing artifacts start as digital representations. This is an issue because it tends to be more difficult to mark up, provide feedback, or gather around digital computer documents. Industry professionals get around this by converting their documents to analog form (i.e. printing) before meetings or group discussions (Gilb & Graham, 1993). However, this practice is often wasteful, time-consuming, and prone to error (Hundhausen, Agrawal, & Ryan, 2010). An obvious solution would be to create software tools that manage social practices within a digital environment (see Hundhausen et al., 2010).

1.1.3.2 Instructor or Institutional Reluctance

Instructor or institutional reluctance to adopt new pedagogies is common. Reasons for this include a perceived lack of payoff in terms of student achievement, loss of classroom control, or lack of time to fully investigate new pedagogies (Lee, 2000; Walker, 2004). Reluctance may also stem from the fact that many interventions are designed for instructors rather than with instructors (Carroll, Chin, & Rosson, 2002). Pedagogies that rely on collaboration also raise widespread fears of cheating (Sheard & Martin, 2011). Regardless of the specific concern, reluctance to adopt new pedagogies is notoriously difficult to change (Walker, 2004), thereby presenting a significant barrier to educational progress. However, this reluctance can be mitigated somewhat through a variety of means. Interventions can be developed in conjunction with instructors, thus transforming them from reluctant participants into champions of the cause (Carroll et al., 2002). Software tools can be developed to reduce any additional burden placed on instructors, and in the case of cheating, can be used as a way to monitor collaboration efforts between students.

1.1.3.3 The "Outside of Class" Problem

Pedagogical interventions are most successful when they are tightly integrated into a given curriculum. Instructors have a huge say in how students must interact during class time. Yet, given that most universities assume a 1:2 or 1:3 ratio between time spent in and out of class, the majority of learning time is often left to individual student discretion. Because socially-oriented pedagogies employ group-based projects, this is somewhat mitigated by the fact that students must continue working in groups outside of class, but many instructors, especially in introductory courses, often prefer individual over group assessment. In these cases, it would be incredibly valuable to provide tools to aid in allowing "student time" to be more social.

1.2 Why do Students Leave Computing Majors?

The previous section reviewed researchers' attempts at identifying the difficult aspects of computing curricula. Implicit in this research is the assumption that the difficulty of computing is what is driving students to leave the major. Rather than identifying difficult aspects of learning to program, one might instead investigate why students choose to leave computing majors. A review of the literature on retention reveals a multitude of reasons why a student might leave the discipline. The following is a summary of commonly cited issues, listed in no particular order:

1. Students do not see themselves as fitting the typical, hacker-like, definition of computing professionals (Allan & Margolis, 2002; Guzdial, Ericson, McKlin, & Engelman, 2012; Ko, 2009; Ruslanov & Yolevich, 2010; Seymour & Hewitt, 1996).
2. Students experience anti-social peers, departments, and/or careers (Cassel, McGettrick, Guzdial, & Roberts, 2007; Ko, 2009).
3. Students cannot connect computing to larger world issues or have little interest in the discipline (Allan & Margolis, 2002; Cassel et al., 2007; Guzdial et al., 2012; Ruslanov & Yolevich, 2010; Seymour & Hewitt, 1996).

4. Students lack confidence (Guzdial et al., 2012) owing to a lack of self-efficacy (Bandura, 1997; Rosson, Carroll, & Sinha, 2011).

5. Students have a hard time making meaningful connections with faculty (Allan & Margolis, 2002; Seymour & Hewitt, 1996).

6. Students have misconceptions about the discipline being more about application usage rather than development (Beaubouef & Mason, 2005).

7. Students are subjected to poorly designed labs that do little to reinforce programming skills (Beaubouef & Mason, 2005).

8. Students are overloaded with too much curriculum. (Guzdial et al., 2012; Seymour & Hewitt, 1996).

Notably absent from this list is that students fail to achieve. Indeed, both Humphreys and Freeland (1992), and Seymour and Hewitt (1996) found that grade point averages between switchers and non-switchers were nearly equivalent [3.0 vs. 3.15 in (Seymour & Hewitt, 1996) and 3.10 vs. 3.07 in (Humphreys & Freeland, 1992)]. While there are some students who will invariably leave the major because of poor grades, this phenomenon is not unique to computing and therefore cannot be the source of the major's higher than average attrition rates. Instead, the list is overwhelmingly composed of issues that possess a social component. This suggests that addressing what makes learning to program hard should focus on remedying social deficiencies in the learning processes traditionally promoted in computing education. My review thus far seems to indicate that socially-oriented pedagogies of engagement offer the best solution path; however, as noted above, pedagogies of engagement often face steep opposition for a variety of reasons. A possible solution, then, might be to adapt the social aspects of these pedagogies so as to make them more palatable to computing educators.

1.3 Thesis
This thesis argues that, in order to address the problem of high attrition in undergraduate computing education, we need to strategically leverage the learning theories that underlie socially-oriented pedagogies of engagement. To do this, I propose the integration of social learning into students’ out-of-class learning activities, which consume a majority of the time students put into computing courses. In computing education, students spend a majority of their out-of-class time solving programming problems in an integrated development environment (IDE), which would appear to be an ideal venue for social learning activities.

This dissertation uses the principles of Situated Learning Theory (Lave & Wenger, 1991), Communities of Practice (Wenger, 1998), and Self-Efficacy (Bandura, 1997) to transform the traditional IDE into a social programming environment (SPE) that allows students to monitor each other’s activities and learn from each other as they complete traditional programming assignments in computing courses. Because an SPE is grounded in the same theories of learning as modern social-oriented pedagogies of engagement, such an approach should produce the same positive benefits in students. Namely, exposing students to an SPE should result in higher grades, retention, and increases in cognitive and affective measures than might otherwise be expected in a traditional course taught without the aid of an SPE. In turn, these gains should result in significantly higher retention rates.

1.4 Motivating Theory
This dissertation uses the theories of Situated Learning Theory, Communities of Practice, and Self-Efficacy as the building blocks of an SPE. This section briefly covers the key principles of these theories, which will be further expanded in the section on related work.
1.4.1 Situated Learning Theory and Communities of Practice

At its heart, any learning theory is attempting to answer the question, "What causes learning?" Whereas some theories attempt to answer this by pointing to specific mental structures or developing abstract models (Mayer, 1981, 2005), Situated Learning Theory (Lave & Wenger, 1991) instead sees learning not as an interaction of structures within a single individual, but as a direct result of social interaction between groups of individuals.

While Situated Learning Theory stipulates that all social interaction results in some form of learning, not all social interaction is equally adept at producing the outcome of learning. Lave and Wenger coin the term Legitimate Peripheral Participation (LPP) to denote social situations in which the main purpose of the social interaction is to produce learning. LPP takes place within a Community of Practice (COP), which is simply a collection of individuals organized around one or more domains (e.g. interests or activities) (Wenger, 1998). LPP, then, describes how learners interact with and learn from a COP. COPs have their own characteristics (Wenger, 1998), but for LPP, a COP must possess three attributes:

1. COPs must provide learners opportunities to participate in core (legitimate) community practices.
2. COPs must allow for advancement in function and additional recognition/status.
3. COPs must maintain an information and/or resource repository that is accessible to learners.

These requirements have basic implications when developing pedagogical tools. In introductory programming courses, learning to program can be considered the core community practice. Therefore, any educational system wanting to implement LPP should provide students with opportunities to participate socially in this practice. This can be accomplished by providing students with opportunities to take on the role of observers, solution designers, discussants, and presenters throughout the course. As students progress through these roles, they take on additional responsibility, thereby providing a path of advancement. Given the nature of the proposed intervention, the third point listed above essentially comes for free. As students use the SPE, a repository of knowledge will begin to build up over the course of the semester. As students encounter new difficulties during the course, they can then reflect on past discussions and interactions to use as resources for solving their current issue.

1.4.2 Self-Efficacy

The term self-efficacy is used to represent a person's belief in his or her ability to succeed at a given task (Bandura, 1997) and has been shown to be a strong predictor in determining success and persistence within an occupation (Bandura, Barbaranelli, Caprara, & Pastorelli, 2001). Self-efficacy is a component of Bandura's (1986) Social Cognitive Theory which states that we can learn from witnessing the experiences of others. In the context of Social Cognitive Theory, self-efficacy can be thought of as the likelihood that a person will learn from a given social experience. According to Bandura, self-efficacy is primarily affected by enactive and vicarious experiences. Enactive experiences are those in which an individual directly participates while vicarious experiences are those in which an individual merely observes. Both types of experiences are externally validated, meaning that an individual's social community helps determine the relative success or failure of a given experience. As an example, Rosson et al. (2011) found self-efficacy to be positively correlated with the strength of an individual's social support. Similarly, Barker (2009) found the level of peer-to-peer interaction to be a strong predictor in determining persistence within computing majors.

When learning new skills, a person's self-efficacy can vary greatly. This is often due to the lack of past enactive and vicarious experiences. If the first few learning attempts go poorly, learners may perceive themselves to be less capable than they actually are in reality. This explains why novices often perceive themselves to be less capable than their peers. Therefore, pedagogies that wish to strengthen self-efficacy should not only provide ample homework opportunities (enactive experience), but encourage students to communicate and be aware of each others' struggles (vicarious experiences and social support). In terms of SPE development, this implies that the system should provide students with the ability to check in on
the progress of their peers as well providing methods for students to communicate about both class and non-class issues.

### 1.5 Research Questions

This thesis poses the following research questions:

**RQ1.** How can we design a *social* programming environment that promotes peripheral awareness, group communication and collaboration, and the ability to check in on the programming state of other classmates?

Given the lack of social features present within the IDEs commonly used by learners, it is a trivial task to make the IDE "more social." The key then becomes identifying specific features that should be made available to students within the IDE. I propose a specific design taxonomy later in this proposal.

**RQ2.** Will the use of an SPE positively impact students' grades, retention, and attitudes?

This question is really the crux of the thesis. Because they are designed for exports, Modern IDEs commonly eschew the possibilities for providing social learning experiences (and learning experiences in general). By introducing social features into the IDE, I aim to leverage social learning theory in order to explore an entirely new way of learning to program.

### 1.6 Proposal Overview

The remainder of this proposal is a beginning attempt to answer the above research questions. Section 2 begins by exploring the design space of social programming (RQ1). My research plan (Section 3) proposes specific measures for capturing student attitudes (RQ 2) and positions me to answer RQ 3. Finally, I conclude the proposal by outlining the current state of SPE development (Section 4) and a literature review of related research (Section 5).
2 The Design Space of Social Programming Environments

Thus far, I have avoided attaching the concept of an SPE to a specific pedagogy that might fit under the umbrella of socially-oriented pedagogies, as I believe the concepts of an SPE to be equally beneficial to all socially-oriented pedagogies. However, at this point, it is necessary to select a single pedagogy in which to ground the SPE. Given the my lab's extensive work in Studio Based Learning (SBL, see Hundhausen, Agarwal, & Trevisan, 2011; Hundhausen et al., In press, 2010; Hundhausen, Agrawal, Fairbrother, & Trevisan, 2009), it makes the most sense for me select SBL as the concrete pedagogical model for the proposed SPE.

SBL is firmly grounded in the theory of Situated Learning, which holds that learning comes about through participation in a social community as learners take on the role of observers, solution designers, discussants, and presenters (Hundhausen et al., In press). SBL can be conceptualized as an iterative process of solution refinement involving two key activities. The first is the design studio, a shared physical space in which students work on assigned problems. While in the design studio, students are peripherally aware of other students' activities. When the need arises, they can become more directly involved by visiting another student's work area, bouncing around ideas and offering help. In the second activity, known as the design crit, students present their evolving solutions for feedback and discussion. These may take place informally with the instructor, or more formally with the entire class and even industry professionals. Given the description above, the question naturally arises of how these activities might be supported by an SPE.

An SPE can do much to support the underlying processes of SBL. Recall that it is in the design studio that students gain an awareness of others activities and have the ability to communicate about problem solving. Therefore, in order to simulate a virtual design studio, an SPE must include both a general awareness mechanism to maintain group awareness and a more specific communication mechanism to be used for direct communication and problem solving. To support SBL's design crit, an SPE needs to support anchored conversations as well as specific pedagogical choices such as group size and author anonymity. This overall design space of an SPE is illustrated in Figure 1 and is this section's subject of discussion.

2.1 Awareness Mechanism

I use the term awareness mechanism to denote a system that allows a user to gather information about other users without the need for direct communication. Phrased differently, an awareness mechanism allows for passive information gathering. In this section, I present two possible solutions that are suitable for an SPE.

2.1.1 Activity-Centered Systems

Popular in social networking sites such as Facebook and Twitter, activity streams aggregate the activity of one or more users into a stream of consciousness for general consumption (see Figure 2). On social networking sites (SNS), individually created activity feed posts tend to be user created and contain thoughts, feelings, or opinions (e.g., "I like ice cream."). However, some feed posts are automatically generated on behalf of the user by other programs (e.g., "John just played Scrabble."). When adapting an activity-centered approach to an SPE, there are many subdimensions to consider, which I describe below.
Figure 1: Design Space of Social Programming
2.1.1.1 Event Selection

A primary concern is the selection of events that compose an individual's activity stream. Whereas traditional activity streams primarily rely on user-generated content, an SPE has the opportunity to supply a large amount of potentially useful feed items generated automatically from user's activity within the IDE. This raises questions regarding the source of a feed posts, the frequency at which feed posts occur, the content of posts that are exposed to the user, and the relatedness of the posts to the current user.

Source. As previously noted, the majority of SNS feed posts are user-generated. Often, automatically generated messages are seen as a nuisance (Facebook allows users to selectively turn off these posts), but the automatic messages generated through SPE activity may be of great interest or use to other students in the class. For example, Student A may find it very helpful to see that Student B encountered the same compiler error. These kinds of vicarious experiences are believed to be vital in the formation of one's self-efficacy, a key factor in determining persistence (Bandura et al., 2001). Therefore, I expect automatically generated messages to play a larger role in an SPE than they do in normal SMSs. However, this gives way to the problem of priority. Given that a single student using an SPE is capable of producing several hundred messages per day, it is likely that user-generated messages will get lost in noise. Therefore, it is likely that user-generated posts need to be placed above those that are automatically generated.

Frequency. The frequency with which automatically generated feed posts are injected into a person's activity stream is a key determinant in the stream's overall helpfulness. Preliminary data analysis of log
data I collected in an introductory programming course in the summer of 2012 indicates that on average, over 7,000 events are recorded on a daily basis. This means that without any filtration, users would receive almost five new activity feed posts per minute. This figure is likely to be too frequent to be useful for users; therefore, determining the appropriate number of new posts to create becomes an important design issue.

Content. The logging feature of my SPE captures a wide variety of data points. Generally speaking, these data points occur while in one of three states: editing, compiling, and debugging. Depending on the course, events that occur in a particular state may be more significant. For example, CS1-type courses, learning how to properly construct code is a key roadblock for many students. In this case, students may be more interested in events that occur in the editing state. However, in more advanced classes, students are confident in their program construction skills and encounter higher level issues. In this case, these students may be more interested in events that occur in the debugging state.

Relatedness. Grudin (1988) states that in order for a collaborative system to succeed, it must be useful to its primary users. In the case of the SPE, the primary users are the students. In order for students of computer programming to find an activity stream useful, it must be capable of providing new insights that were not previously available. For this to occur, it is likely that feed items will somehow need to be relevant to the current user. Finding such relationships can be done in a variety of ways. For example, the SPE might present posts noting a compiler error only if the current user experienced that same error. Another possibility might be to highlight posts to which the user has written a response. Alternatively, relatedness can be user-specified through a series of filters that remove unwanted posts from the stream.

2.1.1.2 Event Composition
The second dimension of activity streams involves the composition of individual feed posts. Composition subdimensions are discussed below.

Access Level. Automatically generated feed posts are created based on another user's input. How much information the system exposes about the event to other users is an open question. For example, in a build event that contains compiler errors, is it sufficient to know just that a student received compiler errors, or should we also indicate which errors were present? Going further, the system could also provide line numbers and even sections of student solutions. Taken to its fullest, events could expose complete documents or the entire solution to other users. While providing deep hooks into the work of others is important for collaborative systems (Erickson & Kellogg, 2002), it also makes it easier for students to plagiarize. Striking a proper balance between providing access to more useful system and facilitating rampant cheating is something that will likely need to be negotiated with individual instructors.

Recommendation System. Tied closely with the relatedness subdimension is the way in which posts are recommended to the current user. I've identified two general strategies that I call the bandwagon and connector strategies. Used by Facebook, the bandwagon strategy highlights the relationship of the active post to the rest of the group. Using this strategy, the recommendation system would display an item like, "Bob and 10 others received the same compiler error." The purpose of this strategy is to highlight popular group issues. In contrast, the connector strategy would attempt to make connections between the active post and the current user. Using this system, the recommendation system would display a message like, "You and Bob both got the same compile error." The purpose here is to highlight potential shared interests or goals between the current user and the rest of the group. Note that these strategies are not mutually exclusive. Such a message might read, "You, Bob, and 5 others got the same compiler error."

Verbosity. While it is completely possible from a technical standpoint to include every data point in each feed post, it is unlikely that this would be the most effective method. Twitter messages have a hard 140 character limit, while Facebook hides content that exceeds a certain length. Both systems prevent a single
feed post from overwhelming the user with information and taking up too much screen space. Likewise, determining an appropriate level of verbosity in an SPE is a key consideration.

2.1.2 User-Centered Awareness System
An alternative to stream-based profiles is the user-centered model. Whereas stream-based profiles promote peripheral awareness through a constant stream of incoming feed posts, a user-centered approach groups individual data points by the users that generate them (Figure 3). With this approach, other users are only aware of each user's most recent action. In doing so, the user-centered approach sidesteps many of the dimensions present in activity-centered systems. Instead, a user-centered approach must concern itself with appropriate states, access level, relatedness, and verbosity.

**States.** Recall that the user-centered approach only displays a user's most recent action. However, given a large number of potential actions, some of which last only a few seconds, this kind of system ends up being unusable, as the constantly changing set of actions is likely to create an unreadable mess. To compensate, it makes sense to group like actions into a set of states that are less likely to change at a rapid pace. This leads to a question regarding the number of possible states. Too few will result in a static interface that gets neglected, whereas too many will run the risk of overwhelming users. Based on a task analysis of computer programming, four types of events are of potential interest: editing, compiling, executing, and debugging. These four states encompass the primary activities of programmers and should provide sufficient awareness to other students. Additionally, to represent a user's status within the SPE, two additional states are needed: online and offline.

**Access Level.** Access level is used here in the same way it was used in the activity-centered approach: namely, the granularity of document access granted by the system. For example, when a user is in the debugging state, the system could grant the user access to a person's breakpoints, stack traces, the active line being debugged, or the entire code document.

**Relatedness.** This dimension refers to the way in which another user's state is related to the current user. For example, the system could highlight states that match the current user's state. Alternatively, states could be grouped and ordered based on user preferences.

**Verbosity.** While a feed post must contain the user's current state, it might also contain other useful information. For example, the state could list the current document being edited or the results of the most recent compile attempt.

2.1.3 Presentation
Presently, the Facebook-style stream-based presentation model is the gold standard for presenting activity-centered content. Yet, this view may be augmented with more detailed views. For example, the system might present individual profiles (Figure 4) that include a single user's activity statistics and a summary of recent activity or a customized timeline visualization (Figure 5) of the user's activity based on a variety of selectors. Perhaps even more so than in the activity-centered subdimension of the same name, alternate presentation modes are of critical importance to a user-centered awareness mechanism. This is because, unlike the activity-centered model, a user-centered awareness mechanism does not have a notion of a history. Therefore, providing views that contain both individual and group histories becomes an important aspect to consider. From a research standpoint, these views raise the issues of content (what to present) and interactivity (manipulability of the view data).
2.1.4 Summary
Both Situated Learning Theory and the construct of self-efficacy place heavy emphasis on peripheral awareness. For Situated Learning, peripheral awareness leads to the development of an active and vibrant community. Self-efficacy sees peripheral awareness as promoting opportunities for vicarious experiences, which are important factors in shaping one’s opinion of one’s own abilities. Therefore, the
goal of any SPE should be the promotion of peripheral awareness by providing users with knowledge of their peer's activity. An activity-centered approach accomplishes this through a continuously-updated activity feed where actions are streamed in real time to the rest of the group. A user-centered approach groups this raw data for each user into states. However, these approaches are not mutually exclusive. It is not hard to imagine a system that offers dual views, or perhaps offers an activity-centered system as a details-oriented overlay to a user-centered view. In all likelihood, this hybrid approach is likely to be the most effective as it offers the widest variety of choice to the user.

2.2 Communication Mechanism
Unlike awareness mechanisms, a communication mechanism facilitates direct communication between two or more users. Communication mechanisms have two dimensions, which are described below.

2.2.1 Mode
In the context of a generalized communication mechanism, mode refers to the rate at which users can transmit and receive messages within the system. The mode can be either synchronous, as is the case with instant messaging, or asynchronous, as is the case with message boards.

2.2.2 Groundedness
I use the term groundedness to denote how anchored the discussion is in the artifact being discussed. For example, a lightly grounded system would have little connection between the conversation and artifact. Examples of lightly grounded systems include AOL Instant Messenger, Facebook Chat, ICQ, and Internet Relay Chat. In contrast, a more highly grounded system might have discussions attached to the artifacts being discussed or might even embed the discussion within the artifact itself (e.g. Suthers & Xu, 2002).

The ability to pose context-specific questions is an important aspect of both SBL's design studio and critical review process. Consider the typical computing course in which an SPE is not present. In order for a student to receive help on a specific code segment, he or she must describe the situation with enough detail and context (i.e. code) such that another person can replicate the issue. An SPE without a review system will likely not fare much better as conversations would still occur outside the authentic programming context. Likewise, in a more formal review process (see Hundhausen, Agrawal, et al., 2009), students are required to create and annotate printouts of their code, which must then be transferred back to its original digital form. Again, such a process would benefit from having reviews take place within the same context in which the coding takes place,. In order to lower the barriers to posing context-specific programming questions, an SPE should support a review system that enables students, instructors, teaching assistants, and mentors to pose and answer programming questions directly within the contexts in which they arise (Figure 6). Highly grounded systems have three sub dimensions.

Annotatability. This dimension refers to the subcomponents of an artifact that can be annotated. In the case of an SPE, the most obvious candidate is the code window. Yet, one can imagine scenarios in which it would useful to be able to annotate a number of other components of an IDE, such as compilation error windows, run-time exception windows, breakpoints in the code, variable watch windows, and program output windows. Furthermore, artifacts generated by the SPE itself, namely activity feed post items, also lend themselves to being annotated.

Granularity. This dimension focuses on the level at which annotations can be made for a given component. For example, assuming that an SPE supports the annotation of code, should such annotations be allowed at the level of the document, function, line, sentence, or word? Likewise, is merely annotating the variable watch window sufficient, or should the system allow annotations of specific items within the watch?

Executability. Finally, since many issues that learners encounter as they learn to program have to do with run-time behavior (Ko, Myers, & Aung, 2004), the executability sub-dimension focuses specifically on
annotations of program state. In order for one to obtain help with a run-time problem, must such annotations exist in a “live” execution environment that allows anyone in the community to execute the code, or are non-executable static snapshots of the IDE at those execution points (akin to screenshots) sufficient?

2.2.3 Summary
While awareness systems are appropriate for promoting peripheral awareness, they do little to support the community and artifact-centered discourse deemed important by Studio Based Learning and Situated Learning Theory. For this, more dedicated communication mechanisms with the dimensions described above become necessary.

2.3 Pedagogy
While the previous two sections discuss how important aspects of Situated Learning Theory and Self Efficacy Theory might be incorporated into an SPE, they do little to consider the practical implications of implementing Studio Based Learning or any other pedagogy based on Situated Learning Theory into the classroom. This section discusses potential issues of privacy, ways in which to construct learner communities, and methods to encourage SPE use by students that may arise when integrating an SPE into a course.

2.3.1 Privacy
A central issue to the creation of an SPE lies in students’ willingness to share personal information with online communities. Given the widespread use of online services that collect personal information, people are becoming increasingly conscious of what they share online (see, e.g. Dwyer, 2011). While there is evidence to suggest that younger generations are more willing to reveal personal information to others (see, e.g. Barnes, 2006; Young & Quan-Haase, 2009), others are suggesting the opposite (see, e.g. Dey, Jelveh, & Ross, 2012). This tension yields the question of how nuanced SPE privacy settings need to be. Three privacy sub dimensions are a) the control learners have over the extent to which they reveal their learning activities and progress, b) the level of access learners have to each others’ learning activities, and c) the level of anonymity provided to members. From the outset, this subdimension will be constrained by U.S. FERPA Law (U.S. Department of Education, 2012), which stipulates that student grades may not be revealed to others without their consent, and that student work may not be revealed to those outside of the class without their consent.

2.3.2 Learning Community Composition
This dimension concerns the people to be included in the community that has access to a given learner’s activities. This includes the size of a given learners’ support community (the entire class, section, or
smaller “programming circles”), how community members are selected (by the instructor, or by the learners themselves), and what mentors are included (including teaching personnel, peer mentors from more advanced courses, and even industry professionals).

2.3.3 Instructor Tools
In order to gain a broad acceptance in the academic community, an SPE must provide some tangible benefit to instructors. Likely, this benefit will be realized in the form of a centralized instructor dashboard. Possible features for this dashboard include providing information on student activities (e.g. common partners, lines of communication, etc.), common stumbling blocks within a programming assignment, possible attempts at cheating, and classroom management tools (e.g. assignment creation, user management features, etc.).

2.3.4 Incentive Mechanism
As the usefulness of social software is based on the rate at which it is used (Grudin, 1988), a key issue of any SPE is the incentive mechanism that will most effectively promote active participation among students. In formal academic settings, instructors can tie participation within the SPE to grades. Alternatively, an SPE might support gamification (Deterding, Dixon, Khaled, & Nacke, 2011) — the inclusion of game-like elements. For example, an SPE might include achievements (see, e.g. Jakobsson, 2011) for accomplishing certain tasks, or be awarded points or levels based on progression (see, e.g. “Reputation Patterns - Design Pattern Library - YDN,” 2012). In this way, gamification can be used to represent community artifacts, which are known to be an important aspect in the development of a community of practice (Lave & Wenger, 1991). These artifacts serve as evidence of membership and/or progression within a community (Wenger, 1998) and are vital in the progression from legitimate peripheral participation to full participation (Lave & Wenger, 1991).

2.3.5 Summary
The pedagogical decisions made when implementing Situated Learning Theory have an immense impact on its overall pedagogical effectiveness. For example, the level of anonymity given to students as they complete reviews is known to affect group effectiveness (Sosik, Avolio, & Kahai, 1997). Likewise, mentorship in the form of moderation, has been found to be an extremely important factor in the effectiveness of pedagogical code reviews (Hundhausen et al., 2011). Therefore, to determine the maximum effectiveness of an SPE, it may be necessary to test the same SPE under different pedagogical conditions.
3 Research Plan

This section describes past and planned research activities related to my dissertation. An overview of these activities is listed in Table 1. To begin, I present the general hypothesis that has been guiding my activities:

H1. Students who actively participate in a social programming environment will score higher on attitudinal measures, exhibit more productive programming strategies, and receive higher scores of achievement than students who either do not use or are not exposed to a social programming environment.

This section outlines the general research plan that I will to employ in order to test the above hypothesis. Note that some of the activities in this plan have already been carried out, some are currently underway, and some will be carried out in future work.

3.1 Study Design

Studies that investigate pedagogical change are more likely to be sensitive to changes if they are carried out over longer time periods (Henderson, Beach, & Finkelstein, 2011). This implies the need for a classroom study. Because classroom enrollment is non-random, the study must therefore be classified as quasi-experimental (Shadish, Cook, & Campbell, 2002). Due to logistical and time issues, it is difficult to continuously measure student attitudes. Therefore, student attitudes will be assessed only at the beginning and end of each condition. However, both achievement and behavior can be collected much more easily, and in these cases it makes sense to investigate these measures at pivotal moments during the course (e.g. exams in the case of achievement, and programming assignments in the case of behavior). Given this rationale, it seems most apt to test the hypothesis using a control vs. treatment group study design that employs pre/post test samples to compare achievement and time series analysis to compare achievement and behavior. The rest of this subsection outlines the participants, materials, and procedure that will be used in the testing of my hypothesis.

3.1.1 Participants

Participants in each condition will be students enrolled in a particular offering of WSU’s introductory programming course (CptS 121). Students enrolled in the Fall 2012 offering will be assigned the control condition, while students enrolled in the Fall 2013 offering will be assigned to the treatment condition. Approximately 200 students are presently enrolled in the Fall 2012 offering. This number is expected to be similar to the number of students in the Fall 2013 offering.

<table>
<thead>
<tr>
<th>Milestone</th>
<th>Target Completion Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design study</td>
<td>Spring 2012</td>
</tr>
<tr>
<td>Build data logging tool</td>
<td>Spring 2012</td>
</tr>
<tr>
<td>Beta test data logging tool</td>
<td>Summer 2012</td>
</tr>
<tr>
<td>Baseline data collection for main study</td>
<td>Fall 2012</td>
</tr>
<tr>
<td>Develop SPE tool</td>
<td>Spring 2013</td>
</tr>
<tr>
<td>Pilot test SPE in classroom setting</td>
<td>Summer 2013</td>
</tr>
<tr>
<td>Treatment condition data collection</td>
<td>Fall 2013</td>
</tr>
<tr>
<td>Data analysis</td>
<td>Spring 2014</td>
</tr>
<tr>
<td>Dissertation Defense</td>
<td>Spring or Summer 2014</td>
</tr>
</tbody>
</table>

Table 1: Proposed Research Plan
3.1.2 Materials
Both conditions will be exposed to the same core CptS 121 curriculum with the same instructor. Likewise, both conditions will take the same pre/post composite online survey consisting of questions from the MSLQ, C++ Self-Efficacy, and Classroom Community scales (Appendix A) and will use the same data logging tool to capture their behavior. Therefore, the only difference between the two conditions will be the presence or absence of the SPE intervention.

3.1.3 Procedure
Usage of the data collection tool and SPE plugin will be made a course requirement by the instructor. A link to the relevant software packages will be made available on the course syllabus. During the first week of lab, students will take the online attitudes survey. In both conditions, the data logging tool will collect usage and behavioral patterns as students work on labs and programming assignments. In the treatment condition, students will also have access to the SPE. In this condition, SPE-specific metrics will also be collected. Finally, during the last lab, both conditions will take the attitudinal post test.

3.2 Independent Variables
Having outlined the basic study design, I now turn to defining the independent variables. As previously noted, the study contains a single intervention: the SPE. However, as illustrated by the review of the design space, an SPE contains several subdimensions. This section details the individual components that will make up the SPE intervention. The selected levels do their best to align with the underlying learning theories, while also taking into account limitations present in both the technical limitations of the IDE that is used in the CptS 121 class (Microsoft Visual Studio 2010), as well as specific instructor concerns (e.g. cheating). Compromises are noted as they occur. All of these choices will be subject to change based on pilot testing that will take place during the summer of 2013. A summary of independent variables is listed in Table 2.

3.2.1 Awareness Mechanism
The SPE will include an activity-centered awareness mechanism. This representation scheme was chosen because it models current popular social networking sites like Facebook, which have been known to be an effective method to present helpful information in a timely fashion (Guy, Ronen, & Raviv, 2011). While an SPE is capable of recording several event types (see Logging Tool Design and Construction for a complete list), only build and debug events will automatically be injected into the activity stream. Additionally, users will have the option of creating their own feed posts in order to ask for help in more specific circumstances. The activity stream will also try to relate and recommend relevant posts. The exact method in which this occurs will be determined later through lab-based usability studies.

According to Situated Learning Theory and the principles of self-efficacy, being aware of peer activities is an important aspect of the learning process. In the context of an SPE, this implies giving students full access to each others' code. However, this notion runs completely counter to traditional university practices, which regard collaborating and viewing the work of others as forms of cheating. Based on interviews with the current CptS 121 instructor, he has similar views. Therefore, the SPE intervention will not allow students full access to others' code documents. As a compromise, the instructor has agreed to allow code viewing as long as it is localized around a given issue (e.g. compile error).

3.2.2 Communication Mechanism
Due to technical limitations, the SPE will not include a highly grounded communication mechanism, but instead will feature medium- and lightly-grounded systems. Recall that groundedness refers to how connected a discussion is to the underlying artifact being discussed. I use the term medially grounded to refer to a system that includes references to an artifact. The medium-grounded system is manifested through the activity stream, which will include small code snippets. In this way, any conversations that occur within a given stream post will retain a limited amount of context. The SPE will also include a
basic chat interface to serve as a lightly grounded, and therefore more generalized, synchronous communication mechanism.

3.2.3 Pedagogy
Because the SPE is targeted for a specific course, there is no need to have robust support for a variety of pedagogies. Along these lines, privacy settings will be nonexistent beyond what is required by FERPA laws. In an effort to reduce the amount of potential noise, students will initially only receive notices from other students in their class section. However, students will have the option to add and remove additional students to their news feed as desired. As noted in the study design, installing the SPE will be a course requirement. It has yet to be determined whether or not the instructor will use the SPE as a basis for participation points within the class. This presents a potential confound as expert moderation is known to be an important factor in determining the usefulness of formalized critical reviews (Hundhausen et al., 2011). In addition, the SPE will include a basic, Stack Overflow-like (“Stack Overflow,” 2012) point system in which students receive points for responding to activity feed posts and having responses marked as helpful (see Figure 7).

3.3 Dependent Variables
I propose a holistic approach to measuring outcomes that attempts to fully explore the impact that an SPE might have on students. The following paragraphs discuss each dependent variable. A summary table of all dependent variables is provided in Table 2.

3.3.1 Achievement
The first and possibly most popular dependent variable is that of student achievement (i.e. grades). It is easy to reason that an effective teaching intervention should elicit an overall improvement in a student's final grade. For additional granularity, the overall grade may be further broken down into individual marks received on homework assignments, quizzes, and exams.

3.3.2 Retention
As this work was motivated by the abnormally high attrition rates common in computing majors, it follows that retention should be measured in any well-designed study. Unfortunately, many students who intend to major in computing remain undeclared at the time of enrollment. Likewise, simply comparing retention between the current course and the next course in the introductory sequence will result in unsatisfactory as the university's CS 1 course contains a sizable amount of students enrolled in other majors who are only required to take CS 1 and not CS 2. As a workaround, I have included questions on the online survey that simply asks students whether or not they intend to major in computing (see Appendix A). Because the survey is taken once at the beginning of the semester and again at the end, it will be possible to identify students who initially intended to major in computing but changed their mind by the end of the course.

3.3.3 Attitudes
Another important dependent measure is attitudinal measures. This is commonly measured through some form of student questionnaire. However, such questionnaires tend to lack external validity. For this reason, it is better to examine standardized measures, and given the social nature of my study, these measures should be ones that are influenced by social behavior. The following attitudinal measures are believed to accurately capture the underlying theories in this thesis.
Self-efficacy (Bandura, 1997) has been shown to be a primary factor in determining persistence within a discipline. Self-efficacy is built upon a person's numerous enactive and vicarious experiences in a particular subject matter. Self-efficacy is known to be low when a person is new to a subject. Self-efficacy will remain low unless that person is allowed to experience enactive and vicarious experiences. An SPE provides ample opportunity for students to be exposed to both types of experiences. Therefore, it is reasonable to expect a rise in self-efficacy after having used an SPE. To measure self-efficacy, I have decided to use both the Motivated Strategies for Learning Questionnaire (MSLQ) (Pintrich, Smith, Garcia, & McKeachie, 1991) and C++ Self-Efficacy Scale (Ramalingam & Weidenbeck, 1998). The MSLQ is widely used and accepted across multiple disciplines, and therefore can be used to compare relative effectiveness of the treatment to other studies. The C++ Self-Efficacy Scale was selected because it most closely matches the C language taught in CptS 121.

Sense of community. An important finding discussed previously is the notion that students feel socially isolated within the major. A major feature of an SPE is to promote social connections and collaborations with other students. Therefore, it stands to reason that using an SPE within a class should promote a higher sense of community than if it were absent. The Classroom Community Scale (Rovai, 2002) was specifically developed to measure sense of community within a classroom and will be used as measure this attitudinal outcome.

Peer learning is an educational practice in which, "Students interact with other students to attain educational goals." (O’Donnell & King, 1999). As an SPE promotes and provides additional avenues for the formation of student work groups, we would again expect a student's sense of peer learning to be higher with an SPE present than it would otherwise. The MSLQ also contains a peer learning scale, which will be used to measure peer learning within the classroom.

3.3.4 Behavior
I use the term behavior to describe a student's problem solving process. In computing, this process mainly takes place on the computer within the IDE. Given the digital nature of my intervention, it is easy to create a detailed history of students' interactions as they complete homework assignments. However, how to best analyze this data remains an open question (Utting, Brown, Kolling., McCall, & Stevens, 2012). One established method of analysis is the Error Quotient (EQ) (Jadud, 2006), which in the past has been demonstrated to be a predictor of overall achievement (Tabano, Rodrigo, & Jadud, 2011). Given
its previous success, I plan to use the EQ as a primary measure of student behavior within the IDE. Furthermore, we can capture the level of SPE involvement as indicated by the number of posts created, the number of responses given, the number of posts marked as helpful by peers, and number of posts whose details were viewed by clicking “See more...”

3.3.5 Cheating
For many educators, cheating remains a top concern (see, e.g. Sheard & Martin, 2011). To address the relationship between an SPE and cheating, the system will collect cut/copy/paste events as well as document “diffs” in order to quantify the extent to which online social environments facilitate cheating.

3.4 Pilot Study: Data Gathering Tool
I had the opportunity to test the data collection tool (see Logging Tool Design and Construction in the following section) in the Summer 2012 session of CptS 121. This study had the following goals:

1. Locate any bugs present in the data collection tool
2. Get a sense of the number and types of logs generated

3.4.1 Locating Software Defects
The summer pilot study proved to be invaluable in locating and eliminating software bugs. Over the course of the summer, I released nine versions of the tool. In doing so, I was able to refine the communication process between client and server, improve client-based data collection procedures, and fix general bugs. Highlights include:

- Eliminating random software crashes
- Expanding the types of events collected. New events added include:
  - Exceptions
  - Cut/copy/paste events
- Added stack traces to debugging events
- Removed Visual-Studio specific (and unnecessary) files from document save events
- Began sending tool crash data to the server

By the end of the summer, the data collection tool was incredibly stable. Version 1.5.0 was released on August 14 and is the version being used in the ongoing Fall 2012 study. To date, there have been no reported crashes or other issues related to the tool.

3.4.2 Characterizing Event Data
The pilot data confirmed the massive amount of information likely to be collected in an actual study. Table 3 and Table 4 list summary statistics from the tools usage in the summer 2012 offering of the CptS 121 course. Given that such a small class was able to create so many events, it is clear that the event selection dimension will be of utmost importance in the final design.

<table>
<thead>
<tr>
<th>Dependent Variables</th>
<th>Independent Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grades (homework, quiz, exam)</td>
<td>User-based awareness mechanism</td>
</tr>
<tr>
<td>Retention (self-report)</td>
<td>Partial code view</td>
</tr>
<tr>
<td>Self-efficacy</td>
<td>Basic chat system</td>
</tr>
<tr>
<td>Sense of community</td>
<td>No anonymity</td>
</tr>
<tr>
<td>Peer learning</td>
<td></td>
</tr>
<tr>
<td>Error Quotient</td>
<td></td>
</tr>
<tr>
<td>SPE usage metrics (clicks, posts, views, etc.)</td>
<td></td>
</tr>
<tr>
<td>Cheating (copy &amp; paste / document diffs)</td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Dependent and Independent Variables
Having developed an SPE, I plan to test its functionality in a manner similar to how I pilot tested the data collection tool in the Summer 2013 offering of CptS 121. The main focus will be on improving stability and eliminating bugs. However, if time and circumstances permit, I may use the full set of materials from the main study. In doing so, I might discover additional insights on design choices that can then be implemented in the Fall 2013 study.

### Table 3: Pilot Data Averages

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of students</td>
<td>20</td>
</tr>
<tr>
<td>Total number of events logged</td>
<td>84,372</td>
</tr>
<tr>
<td>Average number of events per student</td>
<td>4,232</td>
</tr>
<tr>
<td>Average number of events generated per day</td>
<td>1,627</td>
</tr>
<tr>
<td>Average number of events generated per minute</td>
<td>1.13</td>
</tr>
</tbody>
</table>

### Table 4: Pilot Data Totals

<table>
<thead>
<tr>
<th>Event Type</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Debug Event</td>
<td>38,553</td>
</tr>
<tr>
<td>Editor Activity</td>
<td>21,602</td>
</tr>
<tr>
<td>Build Event</td>
<td>12,181</td>
</tr>
<tr>
<td>Save Event</td>
<td>12,036</td>
</tr>
</tbody>
</table>

### 3.5 Pilot Study: SPE

Having developed an SPE, I plan to test its functionality in a manner similar to how I pilot tested the data collection tool in the Summer 2013 offering of CptS 121. The main focus will be on improving stability and eliminating bugs. However, if time and circumstances permit, I may use the full set of materials from the main study. In doing so, I might discover additional insights on design choices that can then be implemented in the Fall 2013 study.
4 SPE Design

The design of the SPE can be broken down into two segments. The first is the underlying data collection tool that will be used to capture student behaviors during the study. The second is a social component that students will use to interact with each other.

4.1 Logging Tool Design and Construction

Any SPE will be limited by the IDE on top of which it is implemented. Therefore, I first began the design of my logging tool by investigating the API exposed by the IDE (in my case, Visual Studio 2010). While the API does expose sufficient functionality for my purposes, it does not provide sufficient functionality to completely explore the SPE design space. In particular, capturing the results of runtime issues is limited only to those that were started with the debugger. Even so, user responses to program prompts is not easily captured, thereby restricting detailed logging of runtime experiences. At the end of my investigation of the IDE's API, the following events appeared to be easily captured:

1. (All Events)
   a. Date of event and
   b. Solution in which the event occurred

2. Build Attempts
   a. List of errors generated during build
   b. List of currently set breakpoints

3. Editor Activity
   a. Cut/copy/paste events
   b. Keystroke activity
   c. Line number on which the edit took place

4. Debugging Activity
   a. Type (start, step over / into / out of, stop)
   b. Current line number
   c. Debug output window

5. Debug Runtime Exception
   a. Type of exception
   b. Exception name
   c. Exception description
   d. Line number
   e. Stack trace

6. Document Saves
   a. The contents of the document

An underline for a particular item indicates that it is used in the calculation of a student's EQ. Note that the majority of items are not used to calculate the EQ. This leaves the door open for the development of more detailed behavioral metrics.

4.1.1 System Architecture Design

The event logging software implements the classic client-server architecture in which each student acts as a client to a web-facing server (see Figure 8 and Figure 9). Upon generating an event, the client attempts to send the event data to the server. For several reasons (server inaccessible, the client does not have Internet, etc.) this send attempt might fail. In this case, the event is saved locally and will be sent along as part of the client's next send attempt. Upon receiving an event, the server attempts to sanitize the data and insert it into the server's event database. The server then sends back the result of this attempt.
Listen for events

Event received?

Sanitize data

Insert event into database

Insert successful?

Tell sender insert not successful

Tell sender insert successful

Event Logging Server Architecture

Application Launch

Events Loop

Event detected?

Can connect to server?

Add current event to list of locally stored events

N

Store event locally

Logs successfully received?

Send locally stored logs to server

N

Y

Application Close

Y

N

Remove successfully sent logs from local storage

Logs successfully received?

Application Launch

Events Loop

Event detected?

Can connect to server?

Add current event to list of locally stored events

Y

Logs successfully received?

Remove successfully sent logs from local storage

Application Close

N

Y

Store event locally

Send locally stored logs to server

Figure 8: Event Logging Client Architecture

Figure 9: Event Logging Server Architecture
4.2 Social Programming Component

The development of the social programming component will follow the user-centered design process (Sharp, Rogers, & Preece, 2007) and will undergo continuous iterative design until the very beginning of the Fall 2013 study. As a result, the ideas presented here are preliminary and are expected to evolve over time.

4.2.1 Activity Stream

The activity stream represents the central information hub within an SPE. As illustrated in Figure 10, it is designed to be open as students work on programming tasks so that they remain aware of others' activities. This is important as it gives students opportunities to form groups around common programming issues. For example, Michael just got a compiler error and notices in the activity stream that someone else also recently got the same error. With that information, he then initiates a dialog with the other student and both work together to solve their mutual problem.

Figure 11 details the individual components of the activity stream. Each feed post has up to five components (Figure 11, items 2-6). All feed posts begin with the posts' author. Clicking on the user's name opens the profile page for that user (Figure 13). Next is the post's message. Automatically generated posts will have simple messages (e.g., "Jack compiled main.c and got a compile error."), followed by how the event relates to the viewing user (e.g., “You and 2 others have gotten this error when compiling solutions to the current assignment.”), the number of comments made in relation to the post, and an option to open the post's details view (Figure 12). Feed post messages are intentionally kept minimalistic. This is done for two reasons. First, C compiler errors are notoriously ambiguous; the same error can account for several programming mistakes. Therefore, it is believed that adding this information will not provide a significant advantage. Second, keeping messages short might encourage deeper exploration of issues. User-created posts follow a similar format, but do not contain the relatedness element. Each feed post also contains a "See More..." link that brings up the details view for that post.

![Figure 10: Activity Stream within the Visual Studio IDE](image-url)
4.2.2 Activity Post Details
Due to the limited screen space available for the main activity stream, it is necessary to direct users to an additional screen that contains the post's complete details. As illustrated in Figure 12, the details page presents all of a given event's information. This includes all errors, code segments, and past comments that relate to the event. It is also the place where users can add new comments to the post. When applicable, individual errors can be selected from the overall list (item #3). Selecting an error brings up the surrounding code associated with the error. In the current design, the SPE exposes the error's line number +/- 3. In Figure 12, the error occurs on line 47, which exposes lines 44 through 50. The optimal line window will need to be determined through user studies.

4.2.3 User Profile View
As previously mentioned, clicking on a person's name will open that person's profile view (Figure 13). In addition to presenting basic usage statistics (item #1), the profile also acts as a hub for receiving personalized messages from other users. While the chat interface (Figure 16) is expected to be the most commonly used component for facilitating discourse among students, providing profile-based message capabilities is intended to be used as a sort of answering machine (i.e. "leave a message") when a user is either not online or unavailable. User-created posts are slightly differently from automatic feed posts in that there is not a “details” view. Instead, communication takes place directly in the profile's activity...
stream. Because user-created messages are likely to get lost in the noise of automatic items, the ability to filter by type (user / automatic) is provided.

### 4.2.4 Group Selection

Automatically subscribing students to all the activities of the entire CptS 121 class is not likely to provide a useful benefit. For this reason, the SPE will have a group selection mechanism by which users can add and remove individuals from their activity stream (Figure 14). By default, students will be subscribed to their course section (approximately 20 students). A student's section represents their core group and cannot be removed by students. Not allowing students to remove others from their section ensures that each student's programming activity is visible to a minimum number of people (approximately 20 students) and prevents the scenario in which a student is not being followed by anyone in the class. However, given that students may have friends in other sections, they will be able to add students from other sections into their group. Note that students will also be automatically subscribed to course instructors and TAs (Christopher Hundhausen in the example) will not be able to "unfollow" them. Furthermore, while it is common for social networking sites to list a user's friend count, I have decided to not include such a feature into the SPE. Advertising friend counts encourages the mass accumulation of friends, thereby increasing the potential signal-to-noise ratio present in the activity stream that may ultimately harm the educational effectiveness of the SPE.
Figure 13: User Profile View

Figure 14: Adding / Removing Users from Group
4.2.5 Asking For Help

Anecdotal evidence collected during previous semesters suggests that students often use Facebook groups to ask other students specific questions about a particular code segment. As it stands, the only code that is inserted into the activity stream is related to compile or runtime errors. However, this system doesn't allow students to ask questions about code that works. While students could paste code directly into the activity stream, I want to provide something more convenient. To this end, I propose the "Ask For Help" feature, illustrated in Figure 15, as a means of asking code-specific questions. To use, students first select the block of code that they have a question about. Next, they right-click on the highlighted code and select "Ask For Help" from the context menu. This then brings up a dialog in which they can provide additional information about their problem. Upon completion, this information then appears in the activity stream, where it will be seen by other students.

Given the course instructor's concern over cheating, it is likely that this feature will have to be regulated. For example, the system could limit the number of times that students can ask for help or instead limit the number of lines that students can ask about (e.g. 100 lines per assignment). Along this same line, the ability to paste content into the activity stream and chat windows may have to be restricted sufficiently enough to discourage mass copy and pasting of code. In the end, the final solution will have to be worked out in cooperation with the course instructor.

Figure 15: Asking for Help
4.2.6 Chat

While the activity stream is great for providing peripheral awareness, it is less useful for facilitating real-time discourse. For this purpose, I've incorporated basic chat functionality into the SPE (Figure 16). In addition to a global, class wide chat, students will also be able to create individual rooms for small group discussion.

Figure 16: Chat Interface
5 Related Literature
The building of an SPE is very much an interdisciplinary activity. The SPE, grounded in learning theory, must draw from previous work in community building, novice programming environments, and programmer aids. The construction of an SPE also raises new issues, the most notable being cheating and personal privacy. This review provides an overview and attempts to situate the SPE inside each of these lines of research.

5.1 Theories of Learning
In my introduction, I state that any learning theory is essentially attempting to answer the question, "What causes learning?" In the process of answering this question, the learning theory also implicitly answers the question, "How can we make learning happen?" The amount that can be written on these two questions, and learning theories in general, would far exceed the length of the rest of this proposal. In the interest of space and time, I instead focus my attention onto the theories of cognitivism and constructivism.

5.1.1 Cognitivism
To a cognitivist, the human mind and learning are described in terms of models and theories. Like a computer, the inner workings of the mind are broken down into a variety of theoretical subsystems whose combination forms the human information processing system (Mayer, 2005). Each subsystem describes certain characteristics and attempts to explain why learning does or does not take place. For example, Mayer's Cognitive Theory of Multimedia Learning (2005) is based on the theories of dual channel (Paivio, 1986), limited capacity (Chandler & Sweller, 1991), and active processing (Mayer, 2009). Often accompanying each subsystem is a list of best practices. As an example, dual channel theory suggests that learning is more likely to occur when information is presented in both visual and auditory form. From this example, we can see that when designing for learning, the cognitivist approach is to structure content in such a way to activate a given model's structures in the most efficient manner. Implicit in this approach is the idea that knowledge is absolute and that an instructor's goal is to simply transmit this knowledge to students. However, knowledge transmission (i.e. lecture) is widely regarded to be an ineffective method of instruction (see, e.g. Smith et al., 2005). Instead, it is argued that instruction should be learner-centered (National Research Council, 2000), which requires a different theory of learning.

5.1.2 Constructivism
Constructivist theories of learning reject the belief that knowledge is absolute entity to be transmitted, and instead believe knowledge is individually constructed by learners (Papert, 1991). Constructivists believe that knowledge is gained as learners, "examine thinking and learning processes; collect, record, and analyze data; formulate and test hypotheses; reflect on previous understandings; and construct their own meaning" (Jonassen, Davidson, Collins, Campbell, & Haag, 1995). Within constructivism exist two main branches: cognitive constructivism, which believes that learning occurs at the level of the individual, and social constructivism, which believes that learning takes place within a social context. The work of Piaget falls into the realm of cognitive constructivism (see, e.g. Gruber & Voneche, 1995) while the works of Vygotsky (1978), Lave and Wenger (1991), and Bandura (1986, 1997) would be considered social constructivism. The aforementioned pedagogies of engagement (see, e.g. Smith et al., 2009, 2005) also fall under the constructivist umbrella. Given the positioning of an SPE as a social learning tool, it makes sense to ground the development of an SPE in the theories of social constructivism.

5.1.3 Selected Theories
As previously mentioned, the design of the SPE has been grounded in Situated Learning Theory (Lave & Wenger, 1991), Communities of Practice (Wenger, 1998), and self-efficacy (Bandura, 1997). This section discusses each of these as well as Studio Based Learning (see Carter & Hundhausen, 2011), a pedagogical implementation of Situated Learning Theory.
Situated Learning Theory believes that learning cannot occur apart from social participation because knowledge is a social construction relative to a particular community of practice. Situated Learning differentiates accidental (i.e. unintended) learning from purposeful learning through the construct of Legitimate Peripheral Participation (LPP). Taking the construct of LPP one word at a time, LPP defines a particular form of participation that is:

- **Legitimate**—Participation must be a means of demonstrating belonging to a community of practice.
- **Peripheral**—Participation is variable. Learning occurs as learners take on multiple and varied roles within a community of practice, ranging from peripheral to central.

Both terms refer to Communities of Practice, which, other than their self-perpetuating nature, can be conceptualized as a normal group of people and have a great power over knowledge and learning. According to Lave and Wenger, a community's standards prioritize certain actions or beliefs over others and adoption of these attitudes demonstrates community membership, and therefore learning. However, while Lave and Wenger do an excellent job of defining how learning occurs, they intentionally, "[focus] attention on the structure of social practice rather than...pedagogy" (p. 113). Therefore, it is necessary to look elsewhere for specific pedagogical recommendations. For this, I turn to Studio Based Learning.

In many ways Studio Based Learning (SBL) can be thought of as formalized apprenticeship. In The Reflective Practitioner, Schon (1983) outlines the four characteristics of SBL:

1. Classroom assignments should be project-based.
2. Student work should be periodically evaluated both formally and informally through design critiques.
3. Similarly, students should be required to engage in critiquing the work of others.
4. Design critiques should revolve around the artifacts typically created by the discipline.

In computing, this usually involves incorporating periodic critical reviews into the normal homework cycle (Carter & Hundhausen, 2011). Using previously defined terminology, this has been more or less an exploration of the "design crit" aspect of SBL. On the other hand, the SPE proposes to explore the informal, "studio" aspect of SBL by facilitating peripheral awareness and group discussion. However, future iterations of the SPE could add support the design crit, similar to the capabilities present in commercial software such as Code Collaborator (SmartBear Software, 2012).

A key to evaluating the effectiveness of an SPE is the concept of self-efficacy. As stated by Albert Bandura (1997), "self-efficacy refers to beliefs in one's capabilities to organize and execute the course of action required to produce given attainments." (p. 3). Stated more plainly, self-efficacy represents a person's belief in his or her ability to succeed at a given task. Self-efficacy is known to be a contributing factor when selecting computing as a major (Papastergiou, 2008) and also plays a role in overall career selection (Bandura et al., 2001). According to Self Efficacy theory, an individual's self-efficacy is obtained from four sources, as described below.

**Enactive experiences** are ones in which the learner is the primary participant. For example, attempting to lift a weight is an enactive experience that will affect a person's weightlifting self-efficacy. As these events are directly experienced, they tend to be the most influential in determining self-efficacy. Early enactive experiences are of critical importance and poor results can lead to significant drops in self-efficacy. Interestingly, the positive or negative impact of an enactive experience is socially determined as most tasks do not have inherit benchmarks for success. Going back to the weightlifting example, if I struggle to lift 25 lbs. and am told that 25 lbs. is incredibly light, my weightlifting self-efficacy is likely to decrease. Conversely, if I am told that 25 lbs. is an astonishing amount of weight to lift, my weightlifting self-efficacy is likely to increase.
Vicarious experiences are those that are not directly undertaken by the learner. Generally, these experiences have a weaker impact on self-efficacy, but vicarious experiences have the power to override poor enactive experiences. Vicarious experiences tend to have more weight when a learner is unfamiliar with a given task and has ill-formed self-efficacy and are most effective when the person being observed is judged to have characteristics that are similar to those of the observer. For example, if I witness someone with my height, weight, and age lift a weight, I will then tend to believe that I too can lift that weight. Conversely, watching a professional weightlifter lift a weight will not likely have an effect on my self-efficacy. Vicarious experiences are especially useful when used as a method for socially validating enactive experiences. For example, assume that I judge myself to have a poor enactive experience when lifting weights. However, I witness someone similar to me struggle when performing the same task. Having witnessed someone else struggle, I may start to think that I'm not doing as bad as I thought. Alternatively, I may start to think that the task is inherently difficult and therefore is less of a reflection of my personal skill.

Verbal persuasion refers to after-the-fact affirmations of faith and encouragement. As such, saying, "good job" would not count as verbal persuasion whereas, "Don't give up, you can do better!" would be counted. According to Bandura, verbal persuasion is likely to result in a boost to self-efficacy as long as the encouragement is within a reasonable distance to a person's ability. Conversely, providing unrealistic encouragement is likely to lower self-efficacy.

Physiological state refers to the physical and mental wellbeing of the person and can affect self-efficacy. For example, a person with a broken arm will probably not believe that he is capable of hitting a home run. Mood also plays an important factor in determining self-efficacy. Bandura (1997) cites research that indicates that being in a poor mood makes it more likely for a person to recall past failures, whereas a good mood makes it more likely to recall past triumphs.

Given that an SPE is intended to be highly social, it would seem that the largest impact that an SPE can have on self-efficacy is providing additional social validation. Recall that computing classes leave many students feeling socially isolated (Ko, 2009) and that infrequent or inaccurate social validation can be harmful to self-efficacy. By allowing students to witness other students struggling and succeeding, an SPE can appropriately situate the struggles encountered when learning how to program. Ideally, an SPE would accomplish this by allowing students to observe the work of others. Unfortunately, as previously discussed, concerns over cheating may prevent instructors from embracing a completely open programming environment in which code is freely shared. However, self-efficacy theory suggests that it may be sufficient to merely inform students others’ struggles and successes. This has been implemented in the proposed SPE by populating the activity stream with key events in other students’ programming processes.

5.2 Learner Communities in Computing Courses

Numerous online learning management systems (LMSs), both commercial (e.g., Blackboard, 2012) and open source (e.g., Dougimas & Taylor, 2003), have been developed to support online learning activities within formal education settings. A central feature of LMSs is their support for computer-mediated communication (see Najafi, Ellis, Cox, & Calvert, 2007). While such communication has traditionally taken the form of synchronous chat and asynchronous threaded discussion, LMSs are beginning to explore communication features that look similar to those of social networking sites. For example, Piazza (Piazza, 2012; Rusli, 2011) enables instructors to create online course communities in which students and instructors can ask and answer questions, track answer progress, and rate answer quality through a social networking-style interface. Aside from its focus on rigorously evaluating the educational effectiveness of learning technologies, this thesis differs from this approach in two ways: (a) it makes learners’ problem-solving activities available to the learning community; and (b) it enables learners to ask and answer questions within the specific problem-solving contexts in which they occur.
5.3 Novice Programming Environments

There exist numerous attempts at making the environment in which students program more conducive to education (see, e.g. Guzdial, 2004; Kelleher & Pausch, 2005). As expected, the design of these systems is usually dictated by what the designers see as the primary educational barrier. Kelleher and Pausch (2005) developed a taxonomy that attempts to situate the motivations behind each system. Using this taxonomy, the work of an SPE would fall underneath the heading of network supported social learning (section 3.1.2). Kelleher and Pausch found only three systems (out of nearly 100) that matched this description and noted in their conclusion that social learning environments was a very promising area for future researchers. Indeed, since the time of publication, several environments that utilize social interaction have been developed. Broadly speaking, these environments either attempt to support traditionally co-located activities (e.g. pair programming) over the Internet or use social data to provide students with additional programming insights. Collabode (Goldman & Miller, 2009, 2011), JavaWIDE (Jenkins et al., 2012), and Saros (Salingger, Oezbek, Beecher, & Schenk, 2010) are all examples of systems that facilitate real-time collaboration over a networked environment. Common features include the ability to co-edit a document, see other documents being edited by other users, and view document edits made by other users. While this line of research does show promise, it ignores the popularity of individual assignments in introductory courses. Because these systems promote the collaborative construction of programming assignments, it becomes difficult for instructors to assign individual credit. On the other hand, the model utilized by an SPE encourages students to collaborate on different solutions to the same assignment. Doing so allows instructors to continue to assign credit at the individual level.

Another approach to making learning more social has been to pair learning environments with ways to share and discuss projects online. For example, the Scratch novice programming environment (Maloney, Resnick, Rusk, Silverman, & Eastmond, 2010) 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 (Brennan, 2009). The proposed SPE expands on this approach by enabling teachers and learners to engage in grounded discussions of not only computer programs, but also the detailed programming processes that generated those programs. HelpMeOut (Hartmann, MacDougall, Brandt, & Klemmer, 2010) and Crowd::Debug (Mujumdar, Hallenbach, Liu, & Hartmann, 2011) are examples of systems that use the programming behavior of other students to provide code recommendations to other students within the class. Both systems record errors encountered by other students as well as the method in which the error was solved. When a student in the future encounters a similar error, the system then presents the student with a list of recommended fixes. Using this strategy, HelpMeOut was able to find useful recommendations 47% of the time while Debug::Code was more successful with a recommendation rate of 57%. While not a novice environment, LemonAid (Chilana, Ko, & Wobbrock, 2012) is a related system that it attempts to provide UI navigation tips based on the experiences of others. LemonAid achieved a much higher success rate of 90%. In contrast with these recommendation systems, the SPE views programming not as a solo activity in which learners search for help when stuck, but rather as a community activity in which learners can see what others are up to, identify others’ activities related to their own, and actively participate in the learning process by asking others within the community for help, and by offering help to others.

5.4 Social Programming Aids

Similar in purpose to social programming environments for novices, social programming aids use social data in an attempt to aid the programming process. However, in contrast to social novice environments, the focus is less on teaching how to program and more on promoting programmer efficiency and/or team collaboration.
Several systems have been developed to better support team collaboration. In many ways, Jazz (Hupfer, Cheng, Ross, & Patterson, 2004) is for professional development what an SPE is for novice education. Jazz promotes "contextual collaboration," or the integration of collaborative elements within the IDE. To this end, Jazz supports anchored conversations and displays a document's edit history. However, the centerpiece of Jazz is the Jazz Band, a profile-based hub that shows each team member's current activity (online, document currently being edited, etc.). The Jazz Band also serves as the launch point for initiating email or chat conversations with other users. FastDash (Biehl, Czerwinski, Smith, & Robertson, 2007) is a similar system that exists outside of the IDE and provides peripheral awareness of team activities. Another approach is to better support team process such as the ability to collaboratively annotate and review computer code in a shared code base (see, e.g. Guzzi, Hattori, Lanza, Pinzger, & van Deursen, 2011; SmartBear Software, 2012). As with real-time collaborative learning environments, these social team systems are intended to be used by programmers as they work on a shared project. An SPE needs to facilitate collaboration across multiple similar projects.

Another use of social tools is to better introduce newcomers to a programming project. TeamTracks (DeLine, Czerwinski, & Robertson, 2005) uses the activities of other programmers to display the popularity and relatedness of code segments. Similarly, CARES (Guzzi & Begel, 2012) allows programmers to view profile information (e.g. job, email, activity) and contact of any user that made recent edits to a given code segment. While the proposed SPE does not have an analog for this kind of social help system, it does provide an avenue for future research opportunities, as these ideas can be adapted for an SPE. For example, instead of highlighting popular code within a single project, an SPE might highlight common problem areas across all student projects (e.g. "50% of students encountered an error when implementing function foo").

5.5 Cheating

The advent of word processors and the Internet have drastically reduced the difficulty of acquiring information and writing documents. Concern over the misuse of these resources for the purposes of cheating or plagiarism is widely reported and discussed in the educational literature (see, e.g. Fontaine, 2012; Mastin, Peszka, & Lilly, 2009). However, while it is easy to think of cheating as a modern phenomenon whose prevalence is ever-increasing, the fact is that students have been cheating for a very long time. In 1911, the Registrar of Stanford wrote, "and the freshman sees the game of cheating going on as almost a matter of course" (Elliot, 1911; quoted in Sheard & Martin, 2011). Surprisingly, studies have found that cheating from online sources is just as likely as cheating from non-digital material (Selwyn, 2008), and according to one study, the amount of online cheating is actually on the decline (Sheard & Martin, 2011). Nevertheless, considerable effort has been spent on plagiarism detection (see, e.g. Fontaine, 2012; Hage, Rademaker, & Vugt, 2011) and concerns about cheating continue to be on the minds of many educators. For example, when presenting a poster on the SPE at a recent conference, I observed that many visitors' first comments were related to cheating. Indeed, the proposed design for the "code view" feature of the SPE has been constrained due to the instructor's concerns over cheating.

By remaining active during students' programming activities, it is possible for the SPE to make two contributions to the literature on cheating. First, by the nature of always on, we will be able to provide more accurate quantifications of the amount of cheating taking place. Second, rather than relying on post-
hoc cheat detection, an SPE can perform better cheat detection through the real-time monitoring of copy and paste events as well as automatically comparing document diffs.

5.6 Privacy

In a recent interview, Facebook (www.facebook.com) CEO Mark Zuckerberg was quoted as saying, "People have really gotten comfortable not only sharing more information and different kinds, but more openly and with more people" (Mike Arrington with Mark Zuckerberg, 2011). This statement fits nicely into a commonly held sentiment that people, especially younger generations, are more okay with revealing personal information online (see, e.g. Barnes, 2006). While it is true that, generally speaking, people are more willing to reveal personal information online versus in-person (Christofides, Muise, & Desmarais, 2009), people also remain cautious about what gets revealed. For example, while most Facebook users provide basic information such as profile pictures, birthdays, and names, users are much less likely to reveal more sensitive information such as phone numbers (R. Gross & Acquisti, 2005; Young & Quan-Haase, 2009). Furthermore, shared information is much more likely to be viewable only by immediate friends (see, e.g. Dey et al., 2012; Young & Quan-Haase, 2009). From this arises an important question: under what circumstances are people willing to reveal their personal information?

Research suggests that, under certain circumstances, people are more willing to reveal personal information. First, people are willing to trade personal information if doing so results in a perceived benefit (Youn, 2005). Another motivator for revealing personal information for increased social capital (Christofides et al., 2009; Ellison, Steinfield, & Lampe, 2007). However, those that are less interested in gaining popularity are less likely to share information for this reason (Christofides et al., 2009). This can be tied back to the notion of perceived benefits. Those that see increased social capital as a benefit are willing to trade privacy. However, those that do not see social capital as a benefit remain unaffected. In the context of an SPE, this might mean that as long as users find the features of an SPE beneficial, they may be willing to expose their programming activities to others. It would also seem that perceived "benefits" are somewhat subjective, and for that reason, it would be helpful for an SPE to provide several benefits in order to appeal to the widest audience. Achievements or a point-based system might appeal best to people more interested in gaining social capital, whereas providing additional opportunities to learn would appeal best to those that are more academically motivated.

5.7 Summary

I began this review by rejecting cognitivist learning theory in favor of social constructivism. Social constructivism believes in the collaborative construction of knowledge, which is reflected in my emphasis on integrating social feature into students' development environments. While there exist several systems that also emphasize social collaboration, they differ from the proposed SPE in the type of collaboration facilitated. While other social learning systems promote the collaborative construction of a single solution, the proposed SPE adopts the unique approach of building an environment in which students work collaboratively on their own, separate solutions. The main advantage to this approach is that it more accurately models how computing is presently taught in the classroom. This allows experimental study to take place directly within a semester-long course, which allows me to fully investigate the long-term impacts of an SPE (e.g. learning, cheating, privacy) on students.
6 Appendix A: Online Survey

6.1 General Demographic Data
1. Name
2. WSU ID Number
3. Age
4. Gender
5. Ethnicity
6. Class Standing
7. Are you presently a computer science major?
   a. (If no) Do you plan to become a computer science major?
      i. (If no) How likely are you to switch your major to computer science?
      ii. (If yes) How likely are you to switch to a major that is not computer science?
   b. (If yes) How likely are you to switch to a major that is not computer science?
8. What is your reason for taking CptS 121?
9. How likely are you to enroll in the next computer science in this sequence (CptS 122)?

6.2 Adapted C++ Self-Efficacy Survey
Rate your confidence in doing the following C programming related tasks using a scale of 1 (not at all confident) to 7 (absolutely confident). If a specific term or task is totally unfamiliar to you, please mark 1.

1. Write syntactically correct C statements.
2. Understand the language structure of C and the usage of the reserved words.
3. Write logically correct blocks of code using C.
4. Write a C program that displays a greetings message.
5. Write a C program that computes the average of three numbers.
6. Write a C program that computes the average of any given number of numbers.
7. Use built-in functions that are available in the various C libraries.
8. Build my own C libraries.
9. Write a small C program given a small problem that is familiar to me.
10. Write a reasonably sized C program that can solve a problem that is only vaguely familiar to me.
11. Write a long and complex C program to solve any given problem as long as the specifications are clearly defined.
12. Organize and design my program in a modular manner.
13. Understand the procedural programming paradigm.
14. Identify the data types in the problem domain and declare, define, and use them.
15. Make use of a pre-written function, given a clearly labeled declaration of the function.
16. Make use of a data structure that is already defined, given a clearly labeled declaration of the data structure.
17. Debug (correct all the errors) a long and complex program that I had written, and make it work.
18. Comprehend a long, complex multi-file program.
19. Complete a programming project if someone showed me how to solve the problem first.
20. Complete a programming project if I had only the language reference manual for help.
21. Complete a programming project if I could call someone for help if I got stuck.
22. Complete a programming project once someone else helped me get started.
23. Complete a programming project if I had a lot of time to complete the program.
24. Complete a programming project if I had just the built-in help facility for assistance.
25. Find ways of overcoming the problem if I got stuck at a point while working on a programming project.
26. Come up with a suitable strategy for a given programming project in a short time.
27. Manage my time efficiently if I had a pressing deadline on a programming project.
28. Mentally trace through the execution of a long, complex, multi-file program given to me.
29. Rewrite lengthy confusing portions of code to be more readable and clear.
30. Find a way to concentrate on my program, even when there were many distractions around me.
31. Find ways of motivating myself to program, even if the problem area was of no interest to me.
32. Write a program that someone else could comprehend and add features to at a later date.

### 6.3 Classroom Community Scale

Below, you will see a series of statements concerning a specific course or program you are presently taking or have recently completed. Read each statement carefully and select the statement that comes closest to indicate how you feel about the course or program. There are no correct or incorrect responses. If you neither agree nor disagree with a statement or are uncertain, select the neutral area. Do not spend too much time on any one statement, but give the response that seems to describe how you feel. Please respond to all items.

1. I feel that students in this course care about each other
2. I feel that I am encouraged to ask questions
3. I feel connected to others in this course
4. I feel that it is hard to get help when I have a question
5. I do not feel a spirit of community
6. I feel that I receive timely feedback
7. I feel that this course is like a family
8. I feel uneasy exposing gaps in my understanding
9. I feel isolated in this course
10. I feel reluctant to speak openly
11. I trust others in this course
12. I feel that this course results in only modest learning
13. I feel that I can rely on others in this course
14. I feel that other students do not help me learn
15. I feel that members of this course depend on me
16. I feel that I am given ample opportunities to learn
17. I feel uncertain about others in this course
18. I feel that my educational needs are not being met
19. I feel confident that others will support me
20. I feel that this course does not promote a desire to learn

### 6.4 Motivated Strategies for Learning Questionnaire

The following questions ask about your motivation for and attitudes about this class. Remember there are no right or wrong answers, just answer as accurately as possible. Use the scale below to answer the questions. If you think the statement is very true of you, circle 7; if a statement is not at all true of you, circle 1. If the statement is more or less true of you, find the number between 1 and 7 that best describes you.

1. I believe I will receive an excellent grade in this class.
2. I'm certain I can understand the most difficult material presented in the readings for this course.
3. I'm confident I can learn the basic concepts taught in this course.
4. I'm confident I can understand the most complex material presented by the instructor in this course.
5. I'm confident I can do an excellent job on the assignments and tests in this course.
6. I expect to do well in this class.
7. I'm certain I can master the skills being taught in this class.
8. Considering the difficulty of this course, the teacher, and my skills, I think I will do well in this class.

The following questions ask about your learning strategies and study skills for this class. Again, there are no right or wrong answers. Answer the questions about how you study in this class as accurately as possible. Use the same scale to answer the remaining questions. If you think the statement is very true of you, circle 7; if a statement is not at all true of you, circle 1. If the statement is more or less true of you, find the number between 1 and 7 that best describes you.

1. When studying for this course, I often try to explain the material to a classmate or friend.
2. I try to work with other students from this class to complete the course assignments.
3. When studying for this course, I often set aside time to discuss course material with a group of students from the class.
Appendix B: Informed Consent

You are being asked to take part in a research study carried out by Dr. Christopher Hundhausen. This form explains the research study and your part in it if you decide to join the study. Please read the form carefully, taking as much time as you need. Ask the researcher to explain anything you don’t understand. You can decide not to join the study. If you join the study, you can change your mind later or quit at any time. There will be no penalty or loss of services or benefits if you decide to not take part in the study or quit later. This study has been approved for human subject participation by the Washington State University Institutional Review Board.

What is this study about?

This research explores the impact of integrated development environment (IDE) features on students’ programming processes and achievement within computer science courses. You are being asked to take part because you are enrolled in the computer science course being considered by this study. Other than the time that it takes to install the IDE plugin, participation in this study should not require any additional time on your behalf. You cannot take part in this study if you are under the age of 18.

What will I be asked to do if I am in this study?

If you take part in the study, your activities within the integrated development (IDE) you are using in this course will be logged. Example events that will be logged include editing operations, file saves, compilation attempts, number and types of errors generated, runtime exceptions, and debugging behavior. You will also be asked to participate in a brief survey at the beginning and end of the semester. Lastly, you agree to release your course grades, including the grades for each course deliverable, to the researchers. All grade information will be kept confidential.

Are there any benefits to me if I am in this study?

There is no direct benefit to you from being in this study. Taking part in this study will allow future educators and researchers to better understand the ways in which IDE activities relate to student problem-solving processes and achievement.

Are there any risks to me if I am in this study?

Your participation in this study will not incur any risks beyond the minimal risks associated with participating in a university course.

Will my information be kept private?

The data for this study will be kept confidential to the extent allowed by federal and state law. No published results will identify you, and your name will not be associated with the findings. We will implement a rigorous procedure in order to ensure that the data we collect are not identifiable within the study. In particular, each course instructor will designate a third party (a colleague or staff member) who will be responsible for generating random codes for each study participant, and for keeping the code list. All data will be associated with those codes, and not your true identity. The instructor and third party will destroy the code list at the end of the semester, making it impossible for the data to be identifiable after the semester is over. The results of this study may be published or presented at professional meetings, but the identities of all research participants will remain anonymous. The data for this study will be kept for 3 years.

Are there any costs or payments for being in this study?

There will be no costs to you for taking part in this study. You will not receive money or any other form of compensation for taking part in this study.

Who can I talk to if I have questions?

If you have questions about this study or the information in this form, please contact Christopher Hundhausen either by email at hundhaus@wsu.edu or by phone at 509-335-3590. If you have questions about your rights as a research participant, or would like to report a concern or complaint about this study, please contact the Washington State...
University Institutional Review Board at (509) 335-3668, or e-mail irb@wsu.edu, or regular mail at: Albrook 205, PO Box 643005, Pullman, WA 99164-3005.

What are my rights as a research study volunteer?

Your participation in this research study is completely voluntary. You may choose not to be a part of this study. There will be no penalty to you if you choose not to take part. You may choose not to answer specific questions or to stop participating at any time.

What does my signature on this consent form mean?

Your signature on this form means that:

- You understand the information given to you in this form
- You have been able to ask the researcher questions and state any concerns
- The researcher has responded to your questions and concerns

Statement of Person Obtaining Informed Consent

I have carefully explained to the person taking part in the study what he or she can expect.

I certify that when this person signs this form, to the best of my knowledge, he or she understands the purpose, procedures, potential benefits, and potential risks of participation.

I also certify that he or she:

- Speaks the language used to explain this research
- Reads well enough to understand this form or, if not, this person is able to hear and understand when the form is read to him or her
- Does not have any problems that could make it hard to understand what it means to take part in this research.
8 References


