EXP: Enabling pedagogical communication between learning and programming environments

We propose to radically advance the technology and processes available to lower-division computer programming courses. Our project focuses on opening students' fine-grained program development efforts to collaboration, assessment, supervision, and introspection while maintaining authentic, project-based tasks and professional, full-featured development environments.

Vision and Goals

In recent years, advanced integrated development environments (IDEs) have moved more fully into the computer science classroom. IDEs are designed to integrate a wide range of tools for development, and therefore be the sole application necessary while focusing on those tasks. IDEs are not designed for educational use, in general, and are missing tools available in typical online learning management systems (LMSs) that support document management, collaboration, assessment, grading and other analysis of student understanding, user authentication, and other educational activities (Rößling, 2008).

Learning benefits from such collaboration activities, enhanced feedback, and frequent, embedded assessments (Bransford *et al.*, 1999; Hoadley, 2004; Linn, in press). However, we believe that many CS courses forgo LMSs and the activities they support. Those that do involve a LMS suffer from poor integration between it and the development environment, forcing students to switch context away from authentic development processes when engaging in collaboration and assessments (Rößling, 2011).

Our project seeks to address these difficulties, offering instructors the ability to deeply embed educational activities within authentic and sustained development practice. We also seek to exploit synergies that arise when combining the strengths of each environment. A central synergy in this project involves opportunities to measure student misconceptions, learning, and behavior. The IDE can serve as a tremendous source of fine-grained data about these characteristics, and combining that information with data collected in the LMS can push frontiers in CS educational research, while providing opportunities for instructors to support their courses and for students to synthesize and manage their learning.

Our detailed goals are the following:

- Designing and implementing the deep integration of learning and development environments through the creation of plugins, server-side programmer interfaces, and libraries. New functionality will include detailed logging of student interactions, and easy and compact access to saved development states for use in collaboration, support, and assessments. We will develop a reference system using popular, open-source systems.
- Development, piloting, and evaluation of learning activities and tools that bridge the strengths of the integrated environments. Examples include (a) collaborative learning activities that integrate and measure developmental practice such as test-driven development; (b) interactive self-assessments that provide feedback to the IDE; and (c) triggering assessments through events in the IDE, such as forcing and measuring self-reflection after multiple failed runs.

- Research into the effects of newly enabled curricular features on aspects of the student program development process. Examples include reflection prompts that respond to repeated errors or to incomplete test suites (Davis & Linn, 2003), and the addressing of weaknesses in aspects of the development process such as positive test bias (Teasley & Leventhal, 1993), debugger use (Ahmadzadeh *et al*., 2005), and timing of test runs (Edwards *et al*., 2009).

We are targeting first and second courses in computer science (referred to as CS1 and CS2 respectively), and will focus our research and evaluation on typical learning outcomes (programming, analysis, computational thinking), the ways that teachers support students through our system, and the ways our system helps students progress and stay engaged.

We envision broadening support for all learners, including students from under-represented populations and those at risk for failure. Primarily, this will be due to the increased feedback and opportunities for self-reflection, better and more frequent opportunities for instructors to support students, and changes in the methods of communication available to students and instructors. One impetus for this project is our experience teaching “lab-centric” courses, in which most of students’ time is spent in a supervised lab with online activities (Titterton *et al.*, 2010), and the success we have had embedding assessments and interactivity into courses at a coarse level. We believe that the advances proposed in this project can lead to even larger learning gains and bigger increases in student engagement than we have seen.

Research Plan and Outcomes

We have years of experience with computer science courses delivered primarily through online lab materials and are well aware of the limitations imposed by separate development and learning environments, have made attempts to provide IDE-like functionality in our LMS. Others are encountering similar problems. Bennedsen and Caspersen (2005) describe the use of video recordings to highlight examples of using an IDE, building a program through incremental development, testing, refactoring, error handling, making sense of the online Java documentation, and object-oriented design through modeling. The videos are necessarily static, however, and interacting with them is obviously not authentic practice. Adcock *et al.* (2007), researching the kinds of pointer-related bugs students encounter, instrument their programming environment to produce log files of pointer errors. Again, no real-time interaction of the programming environment with the LMS is possible.

In this section, we detail our research and development agenda.

Technical development: integrating learning and development environments

The need to create a new learning environment that blends the strengths of LMSs and IDEs underlies the educational, research, and development advances this project promises. Creation of a new environment is a huge task, far exceeding the scope of this project. Fortunately, there are popular applications that are highly modular and open-architected, offering the possibility of incremental development of the functionality we need. Integrating popular applications in this way leverages their wide user and developer bases, their breadth of existing tools, and the future work that will be done with them.

There are several necessary methods of communication between the environments that will need to be supported. The following lists some of the implementation issues:

- Browser events will need to trigger interface changes in the IDE (content, focus, etc).
- IDE events will need to trigger browser events, such as additional browser windows will need to be opened and modified.
- The locking of portions of the system while certain interactions are completed. For instance, editors in the IDE would be locked while a student is assessed or asked to reflect.
- To make context switches between the two systems as efficient as possible, interface elements may need to be shared. For instance, the IDE should show forward and back buttons to allow movement through an LMS-rendered curricular sequence.

Consequently, development will happen in many different areas, including both in the IDE (e.g., plugins, special editors, and perspectives) and the LMS (new modules, activity types, and other features); server-side abstraction layers for the LMS database; and JavaScript libraries, to limit dependence on specific LMS formats.

Logging and displaying implementation behaviors

A LMS can generate a very large amount of data as it supports a course. In our *lab-centric* offerings, most with upwards of 500 separate learning activities in a semester, we have data on student submissions on collaborative activities (e.g., forums), daily quizzes, and homework; timestamps on visits to activities (often more than one million in a semester); and student responses to surveys on demographic and engagement information (Titterton *et al*, 2008). One very large source of data on students has been missing, however: student interactions as they program with the IDE.

We propose to instrument the IDE to log all student actions, including text edits, compilations, test case generation and invocation, and so forth. Data will be stored locally when students work offline and uploaded to a logging service when online, most likely embedded in the LMS.

With logging, instructors will be able to see how students arrive at solutions rather than seeing only the solutions themselves. In our CS 2 course, for example, we advocate *test-driven development*, an approach to program construction in which the following steps are repeated (Beck, 2003):

1. Write a little test that doesn't work, and perhaps doesn't even compile at first.
2. Make the test work quickly, committing whatever sins are necessary in the process.
3. Eliminate all of the duplication created in merely getting the test to work.

A student program submission, however, leaves no clues about whether the programmer took this approach, or instead typed the entire program into a file before running any tests. Modifying the IDE to log the edit/run sequence and the size of the tested program, together with making this log accessible to the LMS, provides information with which an instructor can give students feedback on their development approach. Within introductory programming, Jadud (2006) has had success with this approach on compilation events in the BlueJ Java IDE.

Information about sequences of program runs and tests will also support program understanding activities. For instance, our CS 2 course sometimes includes a *case study*, a worked-out solution to a programming problem. Activities based on the case study include analysis and modification of the accompanying program; logs of what parts of the program each student visited or run during or prior to engaging these activities can substantially enlighten the lab instructor's evaluation of the student's approach to program understanding.

Snapshots of development: collaborating with saved programming states

Every computer science instructor has had to help a student who vaguely refers to a development problem, only to then engage in an iterative process of finding out what the student is actually doing before the problem can be addressed. This basic support would be aided by the ability to save a snapshot of a development process and compactly refer to it. We propose to add a "take snapshot" button to the course IDE, which will provide an id, or URL, that can be included in email and LMS-based communications (e.g., posts in a forum). Any other IDE will then be able to present that snapshot. Storey *et al.* (2006) looked at successful use of such a tool for professional software development.

By allowing annotation of the snapshots—highlights of particular regions with optional textual comments—students and instructors will be able to clarify exactly where confusion lies. Annotations have long been a topic of research in collaborative settings (Cadiz *et al.* 2000; Moreno *et al.*, 2004; Ju *et al.*, 2004).

This tool should provide several advantages for course participants:

- Asynchronous instructor-student interactions would be more efficient: an instructor could quickly know how the student's confusion has arisen and could provide appropriate responses embedded in the same developmental context.
- Students could support each other more efficiently.
- Instructors and students could refer to snapshots within didactic material, from instructor-generated static lab materials to student-generated wiki pages as part of a public project submission or reciprocal teaching activity.
- Instructors could refer to snapshots within online quiz or exam items, allowing students to efficiently (and authentically) see the context of a question.

Prototyping the merged environment

Our initial choices for LMS and IDE are Moodle (moodle.org) and Eclipse (www.eclipse.org). We chose them as our prototype targets because they are popular, open-source, and modular and extensible. Moodle has over 330,000 registered users. Eclipse is also widely used; as early as July 2005, an article in *IEEE Computer*'s "Industry Trends" column proclaimed Eclipse "the Dominant Java IDE", and many programmers prefer it for C++ work as well. The complexity of the Eclipse environment can present challenges for students new to programming. However, there are ways to mitigate these challenges (e.g. Chen & Marx, 2005), and our proposed work will reduce the problem even further.

With the relatively recent success of Moodle and Eclipse, the ability to quickly and efficiently develop a prototype of this type has only recently become feasible. With successful research and learning outcomes, it should be feasible to move beyond the prototype stage and integrate additional LMSs (e.g., Sakai) and IDEs (e.g., BlueJ, Netbeans).

The process of writing plugins for Eclipse is well documented (e.g. D'Anjou *et al*., 2004) and understood. In particular, we expect to build on existing plugins that implement some of the functionality we propose. Examples include the following:

- The DrJava plugin (Reis, 2004) installs an "interaction pane" to allow Java to be used interactively.

- The Gild (Storey, 2003) and Penumbra plugins (Mueller, 2003) simplify the Eclipse menus and file access.
- JExercise (Trætteberg, 2006) administers exercises and evaluates student solutions.
- McKeogh and Exton (2004) describe a "Watcher" plugin that tracks mouse movement, key strokes, scrollbar movement, and document changes, and exports this information to Excel macros that provide visualization facilities.

Technically, integrating a full-featured browser (the container for Moodle) and a Java application (Eclipse) presents some difficulties. We believe that the best solution will involve a web server running in the IDE (Eclipse has the Jetty server built-in) to which the local browser pushes and pulls data. We wish to avoid any solutions that are specific to different types of browsers, such as a browser plugin to handle local sockets.

Curricular development: activities that benefit from merged environments

We propose to develop frameworks for new kinds of learning activities that are appropriate for typical first and second year computer programming courses (CS1 and CS2). We will write instances of these activities, focusing on challenging issues (e.g., array and linked list difficulties) and a broad coverage of topics. Most of the activities will have versions for both the C++ and Java programming languages, languages that covered 96% of first courses in 2001 (McCauley & Manaris, 2002; these percentages have likely changed somewhat). Our activities will be designed to be modular, to ease integration into a variety of instructional settings. As detailed in the Evaluation section, we will measure the success of our curriculum at both universities and community colleges. We are open to testing in high school AP courses, although Eclipse is often considered too advanced for that population.

From the standpoint of the designer of curricular materials, having control over a full-featured IDE is very exciting because of the range of new activities that were previously either difficult or impossible to implement. This section describes the set of new activity types that we will focus on for this project. We will pay special attention to the process of authoring activities of these types, pushing to make the process as easy as possible. In this way we encourage continued (iterative) development as well encourage other instructors to contribute additional activities. We will develop instructor guides to help ensure the best use of the frameworks and materials.

Collaborative learning activities that incorporate developmental practice

One of the main strengths of full featured LMSs is the support they offer for collaborative activities. Students can be formed into groups in various ways, share work, comment on each other's work, and so forth. IDEs, for the most part, support a model of a single programmer coding alone (the incorporation of versioning systems like CVS is an exception). One area for new learning activities, then, is in student collaboration. Industry has particular interest in employees that excel in teamwork and communication skills (Yarnall *et al*, 2007). Simply incorporating student groups from the LMS into the IDE is likely to be a success, with buttons for actions like "Alert my group" for sharing bugs or "Share with my group" for sending code or tests to group members.

Much more is possible, however. Consider an activity that addresses design of comprehensive test suites, called a "mutation contest". For a given program, each student devises a test suite; a collection of buggy programs is generated, either manually or in an automated fashion; and the student whose test suite catches the most bugs wins, preferably with graphics and visualizations that link back to individual tests. These sorts of activities have been implemented manually in courses, but the ease with which they can be included in an LMS-based course is likely to make them much more widespread. Other promising activities involve peer reviews of code, iterative paired programming assignments, and walkthroughs of group code.

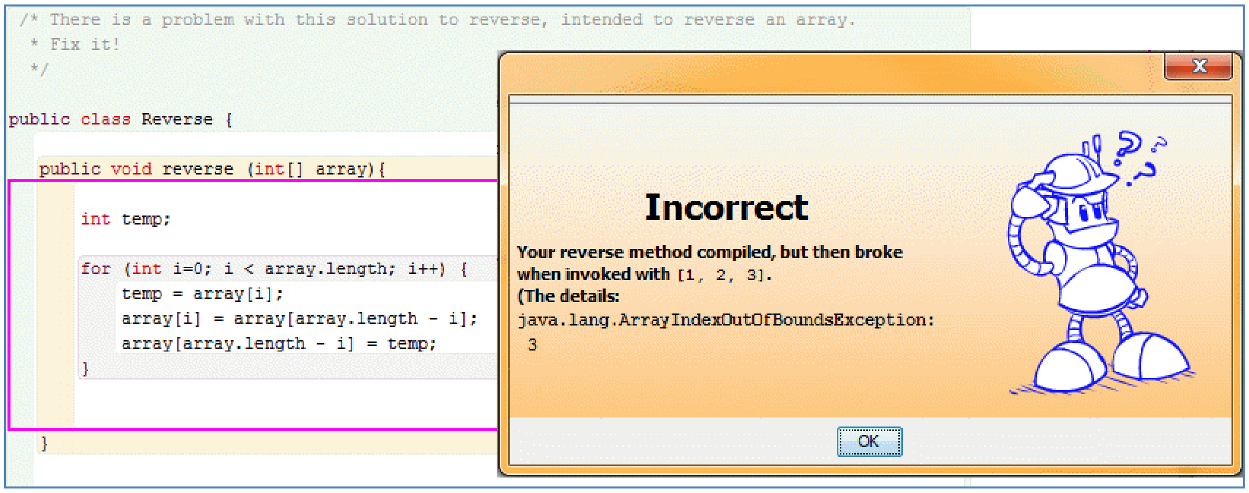
Interactive self-assessments: constrained programming activities with feedback

Our lab-centric courses include an increased number of fine-grained programming activities, including small exercises that address a specific concept currently being taught. In a typical programming course, students engaged in programming tasks receive feedback from people (their instructor and peers) and their programming environment (compiler errors and warnings, run-time problems, test cases when available, and program output). With limited instructor time, the need for feedback falls increasingly on the programming environment. Jadud (2006) details an evocative case in which the feedback from a programming environment (in this case, the Java compiler) does nothing to help an increasingly confused student. Most instructors seem to have similar stories derived from personal experience. Good feedback not only helps resolve the problem at hand, but also improves a student's ability to self-monitor (Corbett & Anderson, 2001). It may be that better and more frequent feedback will differentially help females who are often worse at this type of meta-cognitive skill (Niederle & Vesterlund, 2007).

We thus propose to add a tool to the IDE that supports several kinds of valuable exercises:

- Predict output(s) given particular input(s).
- Identify possible input(s) given a particular output.
- Write the segment (given a description) or complete it (given a partial program segment).
- Assemble the segment from an unordered set of lines of code.
- Generate test cases to be run against unseen buggy segment(s).
- Analyze a segment's runtime or static characteristics.

The tool will be able to evaluate arbitrary source code. It will also constrain types and amount of input that students can create, to focus the student on particular tasks and limit confusion arising



```
/* There is a problem with this solution to reverse, intended to reverse an array.
 * Fix it!
 */

public class Reverse {

    public void reverse (int[] array){

        int temp;

        for (int i=0; i < array.length; i++) {
            temp = array[i];
            array[i] = array[array.length - i];
            array[array.length - i] = temp;
        }

    }
```

Incorrect

Your reverse method compiled, but then broke when invoked with [1, 2, 3].
(The details:
java.lang.ArrayIndexOutOfBoundsException:
 3

Figure 1: A simple BlueJ Exercise activity. The editor is in the background, and the dark outline indicates which portions of the page are editable. The feedback dialog to the right is shown after the student clicks a "test" button, not included in this figure. The feedback is based on the buggy code in the editor.

from syntax errors or changes in non-relevant parts of code snippets. For instance, in an activity focused on debugging tests in conditionals, our tool might present a complete `if` statement and allow editing only between the parentheses following the `if` keyword. For technical reasons, we propose that this tool be developed only for the Java programming language. If a C++ version becomes feasible, we will pursue it—the benefits described here will certainly apply.

***Experiences from Prior Work*:** We have for several years used a similar tool to provide exercises in Scheme (the language used in our introductory programming course). More recently, we have implemented a similar facility—the "BlueJ Exercises" tool—in the BlueJ IDE, and are using it in a newly developed AP CS A course that is being offered for the first time at two sites in the 2010/2011 high school year. Figure 1 shows a screenshot from this tool.

BlueJ Exercises are standard BlueJ projects, run in a slightly augmented BlueJ IDE, that contain one or more Java source files with a specifically authored metadata. When those Java source files are opened, a BlueJ exercise is shown, and students are only able to edit within regions identified by the dark pink rectangle(s). When standard Java source files are opened, the normal BlueJ editor is used. Authoring these exercises is time consuming, because test cases and useful feedback must be thoroughly addressed. We spent considerable effort ensuring that the authoring interface and libraries are easy to use.

Triggering assessments and reflection through events in the IDE

Embedding measurements deeply within a target activity is a foundation of social science research, and has also shown success in helping students learn (Bransford *et al*., 1999; Quellmalz *et al.*, 2000; Tanimoto *et al*., 2002). By triggering LMS activities through IDE events, this functionality is easy to add for activities of interest here: programming, debugging, testing, etc. Building on work such as described in Corbett & Anderson (2001), we will undertake formative research on what types of events and what sort of triggers are appropriate, not overly intrusive, and provide good measurement and/or (small-scale) learning outcomes. It is outside of the scope of this project to implement an intelligent tutoring system (ITS), even though our merged IDE and LMS would provide an ideal system to integrate some ITS activities into a typical programming course. However, we believe that more easily implemented "dumb" triggers (i.e., without use of a cognitive model) can aid student learning (Jadud, 2006).

For instance, consider a student who repeatedly encounters a runtime error, changes a small number of characters in the code, and runs the program again. A student who does this cycle three times in a row is probably unsure about what is causing the error, and would benefit from self-reflection, in the form of hints and/or self-assessments. Our formative evaluations will help specify reasonable triggers and interventions.

Other interventions are possible, since assessment items are but one of the many interactive activities that any LMS provides. When triggered by low-level IDE events, however, these other activities are likely to be overly intrusive (e.g., forcing a student to look over forum posts, or read and contribute to a wiki). Suggesting, rather than forcing, that a student get help, communicate with collaborators in a group project, or take a snapshot of his or her development environment might have beneficial effects without being intrusive.

Timeline

We propose a three-year project beginning in fall 2011 and finishing in spring 2014. There are two distinct development efforts in this proposal: technical and curricular. The technical efforts will begin at the start of the project, progressing from needs assessment and requirements analysis, user studies on early releases, and a release candidate at the end of fall 2012. From there, the technical work will involve bug fixes; new feature development as suggested by users and research findings; and transition to an open-source, community-supported project.

The curricular development efforts will begin in earnest as the first technical milestones are stabilized. The bulk of the curricular work will take place in the 2012 calendar year, with improvements and fixes taking place throughout the 2013 calendar year.

Evaluation of classroom work will begin in fall 2011 with the development of survey and interview instruments, continuing with the collection of baseline data through the 2012 calendar year. Classroom trials with the new tools and curriculum will take place in 2013.

Year 1	**Fall 2011**	Needs assessment and requirements analysis for technical development Begin technical development Target curricular areas identified Begin development of survey and interview instruments
	Spring 2012	Requirements analysis for curricular development Begin curricular development Technical development: beta release Baseline data collected in target courses
Year 2	**Fall 2012**	Technical development: release candidate Formative evaluation of learning tools and curricular activities Baseline data collected within target courses.
	Spring 2013	Evaluation in classrooms. Technical development: iteration, new features, research-based improvements Dissemination: presentation of pilot results (SIGCSE 2013 targeted)
Year 3	**Fall 2013**	Evaluation in classrooms. Technical development: iteration, new features, research-based improvements Transition to community-supported open-source project
	Spring 2014	Dissemination of final results: conferences, workshops, digital libraries. Support of open-source and instructor communities.

Table 1: Timeline of major project milestones

Evaluation Plan

Our internal evaluation plan is focused on formative evaluation, intended to provide the just-in-time feedback necessary to:

1. identify the most salient design needs and user requirements for our proposed tools;
2. design and prototype the right tools in the right way to meet those needs and requirements; and
3. validate or challenge our design decisions for subsequent iterations of the prototype tools.

We will follow a typical user-centered design model of needs assessment, requirement analysis, prototype design and implementation, prototype evaluation, and iteration of the entire process to meet these goals (e.g., Vredenburg *et al*., 2002).

Our development process will begin with needs assessment and requirements analysis. While we know that a need for the proposed IDE features and learning activities exists, a proper exploration of the space of user needs and potential solutions must be completed before design and prototyping begin. Interface designers will working closely with curriculum authors, teachers, and students to understand the opportunities for productive technological intervention (Holtzblatt 1993). Our contextual inquiry procedure will include standard techniques such as semi-structured interviews, classroom observations/rapid ethnography, and think-aloud sessions where potential users will elaborate upon how they perform tasks currently and how their workflow could be improved. We will investigate the potential uses and proper design of the proposed collaborative learning activities, the snapshot functionality, single-page activities, facilities to trigger assessments from IDE events, and the overall integration of an LMS and IDE.

The second stage of our implementation and evaluation process will be the design and creation of prototype solutions that satisfy the needs and requirements. This process will be not only a matter of building solutions, but also an opportunity for early feedback from representative users of the future tools. In this regard, the design and implementation processes will be deeply participatory (see, e.g. Schuler & Namioka, 1993).

The third major stage of our formative evaluation will be running formal user studies in which users try out our prototype solutions, to assess the usability and utility of the tools and curricular activities. The data gained through these evaluations will inform subsequent iterations of our prototype design and development. We will rapidly cycle through new prototype solutions over the course of the project, honing in on optimal solutions through this process of design, implementation, evaluation, and iteration. We will develop a full picture of the successes and failures of our interventions by examining user interactions along many dimensions, including usability, usefulness/utility, learning outcomes, and student engagement. We will measure these factors through semi-structured interviews, surveys, and review of software interaction records.

External Evaluation

The external evaluation will focus on the summative component of the project’s technical and curricular development and will examine both processes and outcomes. The scope of the evaluation will include changes in pedagogical approach; impact of the new tools on student learning, student success, and student engagement; how faculty awareness of student issues and requirements for intervention (support) evolve over time; and changes in communication and behavior among students and between teachers and students. The external evaluation of class-room work will begin in fall 2011 with the development of survey and interview instruments and continue with the collection of baseline data through the 2012 calendar year. Classroom trials with the new tools and curriculum will take place throughout the 2013 calendar year.

There are five high-level complementary outcomes for this project. The first three focus on the student and how they interact with the developed curriculum and tools. The **Student Learning Outcome** (Outcome #1) will use standard learning goals for introductory computer science courses (CS1 and CS2) and assess specific learning goals for each curricular module developed. How and where specifically the new tools help students debug, analyze, and troubleshoot and engage in other aspects of computational thinking will be examined, as well as whether test-driven, integrated development facilitates learner growth in software engineering practices.

The **Overcoming Obstacles Learning Outcome** (Outcome #2) focuses on how and when students get stuck while doing programming tasks. It will be tracked and measured by feedback from the new tools (e.g. instances of interactive self-assessment, the use of triggers indicating the need for instructor help, and increases in teacher-initiated tutoring.) We hypothesize that the increased feedback will help students make steady progress.

The **Student Engagement Outcome** (Outcome #3) will examine student engagement along dimensions such as student enjoyment, time on task, and teacher perceptions of student attitudes. We will study exactly which portions of the tools and curriculum affect these aspects of engagement and why they do so.

The last two outcomes focus on the interactions between teachers and students and among the student population. The **Teacher Knowledge of Student Processes Outcome** (Outcome #4) focuses on increasing teacher knowledge of student learning processes and approaches, student sources of confusion during the learning process, and interim and overall student progress towards course goals. The new tools/curriculum will enable increased data and frequency of reporting, as well as new and effective triggers informing faculty of challenges students have encountered in the learning process. This outcome will also examine how greater teacher knowledge about student progress, or lack thereof, translates to better support of the students and what kinds of support are most successful.

The **Changing Communication Patterns Outcome** (Outcome #5) will involve an in-depth look at teacher-student interactions, as well as student-student (peer) interactions in the course of learning programming. We hypothesize that a change in interactions will result from the snapshot tool and the ability to embed context in posts or emails between learners and faculty. Items of interest include the potential reduction of instructor confusion when reading student posts/emails and the ability to use the newly created tools to better assess the student's path through a complex development practice. Answers to these questions will point to changing and improving patterns of communication between students, their peers, and their teachers.

Data collection methodologies for the external evaluation will include both qualitative data (from interviews and observations with project stakeholders) and short and long-term follow-up survey data. The summative evaluation primarily seeks to measure and document the project's progress toward its goals and the impact of the project as a whole. Baseline project data (both descriptive and comparison data) will be gathered through interviews when feasible. LMS-hosted surveys will be used to track progress along academic milestones both when students are introduced to the new tools/curriculum and later to measure longer term effects and improvements.

It is anticipated this project will impact close to 1000 students across at least four sites—we currently have implementers at U.C. Berkeley, Duke, University of Illinois, and City College of San Francisco, described in the Team section below; we are likely to seek additional presence in community colleges.

In addition to following students, participating faculty at these sites will be interviewed to describe how their expanded level of engagement in the program has affected their teaching and their students. Both students and faculty will be asked for recommendations for program improvement. Through analysis of project and learner outcomes, we will be able to look at learning across different student groups. Through surveys, we will gather self-reports about tool usage, preferences, and collaboration and compare those reports to data we've gathered through our usage logs. Finally, through comparisons between semesters, we will elucidate whether and how our integrated development/learning environment is beneficial for students and instructors.

Dissemination Plan

We plan to ensure the dissemination of this work in a variety of ways. First, we will take advantage of the existing communities around our LMS and IDE applications. A prime reason for choosing Moodle and Eclipse as our prototype targets is their large communities. Moodle has over 500,000 registered users. Eclipse, as noted previously, also is the focus of a large programming community. Both systems are ranked at or near the top in adoption within their category. Moodle has an active CS education community, and we expect that candidates for early adoption of our system will arise from that community. Additionally, both systems have active developer communities, likely to provide support during our design and implementation phases. We will contribute our tools to the open-source repositories maintained by the developer communities of each system, and take pains to support developers that take an interest in our tools.

Second, we will work to place our curriculum and information about our tools in digital libraries (e.g., Computing and Information Technology Interactive Digital Educational Library and the Ensemble NSDL Pathway). This will include careful efforts to tag with proper metadata.

Third, we will place a high priority on the design of well-formed libraries and programmer interfaces used in the prototype integration. This will to help smooth the inclusion of additional systems (the BlueJ and NetBeans IDEs, and the Sakai LMS). Furthermore, there is nothing particularly special about computer science. Many disciplines can point to a widely used, professional, sometimes open-source application that is used in educational settings and that suffers from lack of integration with LMSs. Statistics is a prime example, with the R system.

Fourth, we will present our research results and tool development in conference presentations and publications. We are targeting SIGCSE 2013 for presentation of our pilot work, and SIGCSE and other conferences and journals in 2014 with full results. Lastly, perhaps most importantly, we will leverage the lab-centric communities and instructor support and dissemination tools we have built for our NSF-funded CPATH project. Although the work proposed herein does not depend on the lab-centric format, it does fit nicely.

Prior Work: Lab-centric instruction and the UC-WISE project

Our work in the UC-WISE project (University of California Web-based Instruction in Science and Engineering) provides much of the impetus for this current proposal: our desire for better information about what students are doing in their IDEs, as well as our efforts to bring the IDE into the browser, stemmed from work in the UC-WISE project. The project's goal is twofold:

- to provide technology and curricula for laboratory-centric higher education courses that incorporate online facilities for collaboration, inquiry learning, and assessment, and to investigate the most effective ways of integrating this technology into our courses;
- to allow instructors to customize courses, prototype new course elements, and collect review comments from experienced course developers.

The UC-WISE system includes a database of annotated learning objects, served by linking to the WISE learning environment developed in the School of Education (Linn *et al*., 2004), plus portals into the database for students, instructors, and curriculum developers. Activities provided by WISE and used in a UC-WISE curriculum include online discussions, programming exercises, reading of Web-delivered text, reflection notes, journal entries, quizzes, scripted assessments, and "gated collaborations" where students critique their peers' responses to a seed topic.

At Berkeley, we have implemented lab-centric versions of several lower-division computer science courses that trade lecture and recitation time for more hands-on lab work (Clancy *et al*., 2003; Titterton & Clancy, 2007). These courses have numerous educational benefits; in particular, instructors may view some student work (e.g., quiz responses and collaboration activities) in real time, and intervene if necessary to provide targeting tutoring or other pedagogical support. Student engagement is high, with the majority of students rating the lab-centric format highly.

We have noticed a number of advantages that our courses present to students who historically have been underrepresented in computer science. Our courses provide a much wider variety of activities than their traditionally taught counterparts, allowing more students to shine and supporting a wider variety of learning styles and cognitive abilities. Shy students are encouraged to contribute in collaborative exercises, and receive help without seeking it out. Students who might otherwise drop out are kept in the course with the support provided by staff and the learning environment. See Titterton *et al.* (2010) for a review. To the extent that our proposed efforts expand the use of LMSs, more students from underrepresented populations would share in these benefits.

The UC-WISE project has been the beneficiary of two NSF grants. The first, titled "Online Curricula for Monitored, Closed-lab First-year CS courses" (DUE-CCLI 0443121; $249,703; May 2005–April 2009), was for the development of curricula for lab-centric offerings of an introductory Java programming course and a Java-based data structures course. The introductory course has been offered twice, and is currently undergoing refinement. The data structures course has been offered five times, and three other courses have borrowed substantially from its materials. Titterton *et al.* (2008) discuss one semester of the course, comparing it to a traditionally formatted (non-lab-centric) version of the course taught by a different instructor. Some evidence is shown that supports better outcomes for females in the lab-centric course (on project scores, but not exam scores), that the lab-centric course has a better classroom climate, and that the workload for the two courses was similar. Another publication arising from work on this grant is Titterton and Clancy (2007).

The other NSF grant, titled "CPATH-CB: A community for lab-centric computer science instruction" (CNS-CPATH 0722339; $369,760; August 2007–July 2010), is a recent grant supporting the formation of a community around lab-centric instruction. This effort has several components: evaluation of the current state of lab instruction in CS courses; curriculum development; and tools for authoring, collaboration, and course reviews. The work on this grant is ongoing, and is the focus of several projects undertaken by volunteer undergraduates as well as directly contributing to graduate student Andy Carle's dissertation work.

We are currently in the process of moving the UC-WISE system to Moodle, which, because of its stability and complete feature set, removes the need for us to maintain portal and administrative features. This move will allow us to focus on curriculum development, development of new activity types, and learning research. Work continues in adding functionality to Moodle to make a system that can support all of the activities from the UC-WISE courses.

Intellectual Teams

Management Team

Michael Clancy is a Senior Lecturer in the EECS Department at University of California, Berkeley. Clancy is one of the leading CS education researchers in the U.S., and his experience spans a wide variety of pedagogy: self-paced instruction, labs, case studies, pair programming, peer instruction, and lab-centric courses. With Marcia Linn, Clancy devised online learning environments for novice programmers learning LISP in a project funded by NSF. More recently, he has led the UC-WISE project (also partially NSF-supported), which aims to provide technology and curricula for laboratory-centric higher education courses.

In this project, Clancy will lead the curriculum design efforts and support the technical development efforts.

Dr. Nathaniel Titterton is a research specialist in the EECS Department at University of California, Berkeley. He received his doctorate at U.C. Berkeley in Education, focusing on instructional technology and statistical understanding. His graduate work included research on learning management and content management systems. He taught the lab-centric version of U.C. Berkeley's introductory programming course for four years, led the design of the original lab-centric LMS (UC-WISE), designed the BlueJ Exercise tool, and supervises undergraduates getting credit for evaluation, design, and implementation projects.

In this project, Dr. Titterton will be overall project manager and will be the lead designer and developer of the technical development tasks. He will also supervise the volunteer and for-course-credit undergraduates working on this project (likely to be 8-12 per year).

Andy Carle is a PhD candidate in the EECS Department at University of California, Berkeley. His graduate work is in human-computer interaction, with a focus on instructional technology, design methodology, and qualitative evaluation techniques. He is a senior fellow of the Berkeley Institute of Design and is an active member of the SIGCHI and SIGCSE communities. Andy is the lead designer on pact, an NSF CPATH-CB funded project that facilitates curriculum development and community interaction around lab-centric instruction.

In this project, Andy will coordinate the user-centered design process and take the lead on the qualitative aspects of formative evaluation. He will monitor the usability, utility, and accessibility aspects of the design and implementation of all technological interventions, and will assist in the development tasks.

Advisory and Implementation Team

We have assembled a wide-ranging and extremely experienced advisory team. We will meet as a group twice a year, and engage members in subgroups as necessary. The advisory team will assist in all project phases, but are expected to have the following special roles:

- Learning and engagement: Marcia Linn
- Technical development: Guido Rößling, Thomas Naps

- Curricular development and management of local implementation: Owen Astrachan, Cinda Heeren, Craig Persiko

Owen Astrachan is the Director of Undergraduate Studies in Computer Science at Duke where he has taught for more than twenty years. Astrachan builds curricula and approaches to teaching computer science. This includes an NSF-sponsored, apprentice-learning approach between Duke, Appalachian State, and North Carolina Central and an NSF CAREER Award to incorporate Design Patterns in courses. He was involved early in AP Computer Science: as teacher, as member of the development committee, and as the Chief Reader. He is the PI on the CS Principles project to create a broader, more accessible AP course in computer science. He has been awarded several university-level teaching awards. In 2007 he was an inaugural recipient of the NSF/CISE Distinguished Education Fellow award.

Marcia C. Linn is professor of development and cognition in the Graduate School of Education at the University of California, Berkeley. She is a member of the National Academy of Education and a Fellow of the American Association for the Advancement of Science, the American Psychological Association, and the Association for Psychological Science. She has served as Chair of the AAAS Education Section and as President of the International Society of the Learning Sciences. Board service includes the American Association for the Advancement of Science board, the Graduate Record Examination Board of the Educational Testing Service, the McDonnell Foundation Cognitive Studies in Education Practice board, and the Education and Human Resources Directorate at the National Science Foundation. She has twice been a fellow at the Center for Advanced Study in Behavioral Sciences. Linn has authored several books on teaching and technology.

Guido Rößling received his Diploma in Computer Science from the Technische Universität Darmstadt (Germany) in 1996, and his Ph.D. in Computer Science from the University of Siegen (Germany) in 2002. Since November 2001, he has been a research assistant and senior lecturer at the Technische Universität Darmstadt (Germany). His teaching activities center around usability, e-learning and teaching the fundamentals of Computer Science. His research centers on e-learning aspects, with a focus on algorithm visualization and learning platforms. He currently has around 100 publications on these topics. Since 2002, he has chaired several international Working Groups with a focus on improving the use of algorithm visualizations for teaching and on adapting learning platforms such as Moodle. He has also co-chaired or chaired the Program Visualization Workshop in 2006, 2008, and 2011.

Thomas Naps, currently a Professor of Computer Science at the University of Wisconsin-Oshkosh, earned his PhD in Mathematics from University of Notre Dame in 1975 and has been teaching Computer Science ever since. Since 1987 he has designed numerous visualizations of algorithms to help his students, and is a primary developer of the GAIGS and JHAVE (Java-Hosted Algorithm Visualization Environment) AV systems. He has written numerous refereed papers in the area of AV, conducted workshops and tutorials at SIGCSE and ITiCSE, and chaired six international working groups on visualization at ITiCSE conferences. Over sixty faculty members at other institutions have used his AV systems.

Cinda Heeren is a lecturer in the Computer Science Department at the University of Illinois, where she is primarily responsible for helping 300 students per semester find great joy in mastering the expectations of a rigorous CS2 course. In addition, she recently developed a CS1 course for non-majors employing a strictly lab-centric approach (administered via Moodle). Dr.

Heeren's research projects include leading development of Collaboration Modeling Toolkit to analyze, visualize, and synthesize the program similarity; developing a Discrete Math Concept Inventory designed to aid instructors in evaluating the effect of pedagogical innovation; development and deployment of curricular materials on parallel and distributed computing, in support of the NSF/IEEE - TCPP Curriculum Initiative; and investigating the utility of a remote student feedback system using a small number of tablet PCs in a large lecture hall. She is the faculty advisor to the Women in Computer Science student organization.

Craig Persiko is a tenured Computer Science instructor at City College of San Francisco. He has been teaching CS1 courses there since 1999, as well as CS2 and courses in Software Engineering, XML, programming with databases, etc. He is also the department's advisor to students. Persiko will likely be the Computer Science Department Chair starting fall, 2011.

Through his work with MPICT (the Mid-Pacific Information and Communications Technologies Center, an NSF-ATE Center funded by grant number DUE 0802284), Persiko has been exposed to lab-centric approaches to teaching. He has adjusted his own teaching style in response, to incorporate more in-class exercises in place of lecture time. He has focused on the need to engage and excite students to increase retention, especially among women and minorities.

External Evaluation Team

We are lucky to have an experienced external team to ensure the best possible instrument design, data collection, and analysis.

Dr. Sandra Mikolaski is an educational consultant, leader, and mentor to nonprofit organizations that advocate learning, education, and literacy. She provides consulting services in grant research, grant planning and writing, and grant evaluation, and offers support services such as project management, facilitation, and dissemination. She has an extensive background in educational research and development; program and project management; instructional design and delivery; and skills identification and alignment, including 18 years at Bellevue College where she was the PI for numerous NSF projects and other federally funded grants. Her areas of interest include innovative STEM education; learning communities and collaborative work; evaluating learner success; faculty professional development; and industry engagement.

Dr. Manjari Wijenaike is an educational consultant for grant strategy development and evaluation with over 15 years of experience in STEM education programs and initiatives to encourage underrepresented populations in STEM careers. She works with educational and non-profit organizations to research, write, edit and submit grant proposals as well as implement and evaluate ongoing projects. Dr. Wijenaike has served as PI, Co-PI, and Senior Personnel on 10 NSF grants at Bellevue College. She currently serves on the Evaluative Advisory Panel for two NSF grants being conducted by SRI International. Dr. Wijenaike holds a PhD in cultural anthropology and has extensive expertise in the areas of designing and carrying out qualitative research projects, conducting ethnographic fieldwork, and collecting and analyzing data from focus groups, surveys, and interviews.