Designing a SCALE-UP Style Instructional Computer Lab

Scott D. Kerlin
Department of Computer Science
University of North Dakota
Grand Forks, ND 58202
skerlin@cs.und.edu

Abstract

It is known that traditional lecture is a very passive student experience and online courses/programs now can provide high quality lecture experiences via the Internet. Therefore, the question that should be posed is how to ensure that on-site classroom time can provide the most value for students, encouraging retention and lowering absenteeism. A pedagogy that has shown great potential in this area for other STEM areas is Student-Centered Activities for Large Enrollment Undergraduate Programs (SCALE-UP). The SCALE-UP pedagogy calls for establishing a highly collaborative, hands-on, computer-rich, and interactive environment. This would seem to be a good fit for Computer Science as actively engaging students with programing during all class sessions should increase their success and consequently improve retention. Additionally, such an environment should lower absenteeism since students are no longer passively sitting there during class, but actively participating in the class.

SCALE-UP pedagogy requires a proper environment in which to flourish, the design of the room is very important. Additionally, Computer Science instruction has requirements which are unique from other STEM disciplines which use this pedagogy (such as every student needs a computer, not just every group of students, and a requirement for seamless passing of code fragments between all group members in addition to simple screen sharing). This paper discusses how decisions in the design of a SCALE-UP Style Instructional Computer Lab were made at the University of North Dakota Computer Science Department. The software design considerations and challenges are addressed through a combination off-the-shelf components and in-house projects. Included in these in-house projects is software that can be developed by the students themselves to increase the students ownership stake in the room itself and to the courses presented inside of it. Simulations suggesting traffic/flow patterns for the room are presented to address the physical and hardware design considerations, highlighting the importance of convenient and plentiful workspace for non-computer based work (whiteboards for planning and organization) as well as communication between group members, other groups and the instructor.

1 Introduction

It is known that traditional lecture is a very passive student experience and online courses/programs now can provide high quality lecture experiences via the Internet. Therefore, the question that should be posed is how to ensure that on-site classroom time can provide the most value for students, encouraging retention and lowering absenteeism. A pedagogy that has shown great potential in this area for other STEM areas is Student-Centered Activities for Large Enrollment Undergraduate Programs (SCALE-UP)[2]. The SCALE-UP pedagogy calls for establishing a highly collaborative, hands-on, computer-rich, and interactive environment. This would seem to be a good fit for Computer Science as actively engaging students with programing during all class sessions should increase their success and consequently improve retention. Additionally, such an environment should lower absenteeism since students are no longer passively sitting there during class, but actively participating in the class.

SCALE-UP pedagogy requires a proper environment in which to flourish, as such, the design of the room is very important. Additionally, Computer Science instruction has requirements which are unique from other STEM disciplines which use this pedagogy. It is the focus of the paper to record how decisions in the design of a SCALE-UP Style Instructional Computer Lab (to be referred to as a SUSIC Lab from this point on) were made at the University of North Dakota Computer Science Department. The first section will discuss design decisions for the physical aspects of the SUSIC Lab as they relate to SCALE-UP and why our SUSIC Lab has some differences based on unique Computer Science requirements. The second section will focus on software design choices and decisions that were made. Again, this discussion will include a comparison with traditional SCALE-UP software selections with the choices made in the construction of the SUSIC Lab. The third section of the paper will focus on student/instructor interaction in this new SUSIC Lab, and how the learning within this room is expected to occur.

2 Physical Design

In SCALE-UP, the room itself is often considered one of the most important aspects to the success of the pedagogy[2], as such, the design of any SCALE-UP inspired room needs to be conducive to the learning style of the pedagogy. The main physical components of a SCALE-UP room are: tables, drawing surfaces, computers and viewing screens[6]. Tables must be designed to foster interaction while still providing sufficient space for working. Drawing surfaces provide places for visual learning, ordering, problem solving and presentation. Computers provide research, presentation and activity hubs. Viewing screens permit the entire class to be exposed to the same images and at the same time. Each of these must be considered carefully and implemented in a way that makes sense to the discipline. When a room is serving dual purposes, this is especially important as the preservation of flexibility is required.

The SCALE-UP pedagogy is all about communication and the fostering of interaction and learning as students work together to solve ill-defined problems. As the table is the central vehicle through which this co-operative learning takes place, let us begin with considerations with respect to the tables the students shall be seated at. Traditional SCALE-UP pedagogy utilizes 7-foot diameter round tables capable of seating 9 students each to provide a conducive conversation zone while not feeling crowded[6]. The 9 student capacity allows for division of the table into 3 equal ”pods” of 3 students for smaller group work within the context of the larger 9 person table and provides a support mechanism for students outside of class[7]. While this 3x3 setup is ideal for SCALE-UP, it breaks down for Computer Science when attempting to consider maximum flexibility and dual use purposes for an instructional lab.

Since it is impractical to devote an entire lab to the single purpose of SCALE-UP exclusive teaching when even within the same course different pedagogical approaches may be beneficial at different times, we have to also consider the flexibility of this space, and therefore of the tables we are utilizing within the space. One pedagogy that has a track record of success (under specific conditions) in both education and the professional workplace within Computer Science is Paired Programming[10], one of the off-shoots of Extreme Programming (XP). As the name implies, paired programming focuses on 2 person development teams. Much like SCALE-UP an environment that is conducive to paired programming is very important the success of the pedagogy model [5]. Another consideration that team based programming has higher programmer satisfaction and fewer bugs [11], but too large of groups will reverse those group based benefits significantly[3].

With these different demands and insights into the unique Computer Science STEM discipline, both in education and in the professional workplace, tables which allow for the seating of 6 students was determined to be an ideal number for a SUSIC Lab. A size of 6 permits the instructor to break each table into 3 pairs for small projects and 2 3-person teams for medium projects while still providing a large enough base from which each table may synthesize/actualize larger results (more complex programs, designs, models and testing). Tables of size 8 or greater would cause too much fracturing when working as pairs toward a table-wide goal, while smaller sizes wouldn't result in large enough table groups[6]. A 5 foot buffer zone between tables provides for plenty of room for movement between tables and permits students to move around as needed.

The freedom of movement supplied by the buffer zones between tables is important as students spend a great deal of time moving about the room. They do this to either access their own group's, team's or partnership's whiteboard area or to get a better view of the whiteboard work of other students'. All of the SUSIC Lab's walls have large whiteboard surfaces installed on them, this provides ample drawing surfaces for organizational activities. As Computer Scientists, we learn early on the importance of whiteboards for design and pseudo-coding, and by having whiteboard space for all students and activities based around whiteboard usage, we can instill the practice of drawing things out first and programming second in our students.

The computer component of the SUSIC Lab represents a small departure from traditional

SCALE-UP space. In a traditional SCALE-UP environment, only 1 computer would be provided for each 3 person pod. As Computer Scientists, this is inadequate to the needs of our discipline. In addition to the activity instructions, research and presentation tool that traditional SCALE-UP courses utilize computers for, our discipline also requires the computers for modeling and programming tasks. As such SUSIC Labs should ensure a computer for each student. This can be managed either via docking stations for laptops or by setting up desktop systems at the tables. If choosing the former, mounting of the towers BELOW the table surface is important to minimize desk clutter and improve sight lines and communication between table members. This setup also permits the SUSIC Lab to function as a computer lab during off hours so that students can continue to work on their projects when classes are concluded for the day.

Viewing screens of some sort are also needed so that instructors can present material to all students at once and student generated material may be easily presented the class as a whole. It is important to ensure that these screens (whether wall mounted monitors, televisions, or projector screens) do not interfere with whiteboard access. UND's SUSIC Lab ran into this problem when it's projector screens failed to retract when no longer needed, blocking access to several whiteboards. If using automated or mechanically retraction of screens, a manual method of retraction should be considered in case of failure.

3 Software Design

Much of the SCALE-UP pedagogy relies on students actively investigating ponderables and working with tangibles (analysis, experimentations or simulations)[6]. This leads to a high degree of interactivity between the student and the rest of the class. To ensure that there are no groups which are just going to sit and wait for the instructor to bail them out by providing an answer, instructor tools for creating the required interaction between the different students in the class at large are needed. These tool create the situation where positive peer pressure is the enforcement tool to keep students on track with their tasks and motivated[2]. There also needs to be lower stakes methods of answering questions and "being wrong" where the students can make mistakes and take risks without the worry of what their fellow students will think. Part of this comes from setting a positive atmosphere, and is thus beyond the scope of this paper, but part of this going to come from the communication mediums provided by the room itself. As all students in a SUSIC Lab will have their own computers, instructors can use message boards, forums, blogs, instant messaging and related systems to permit students to communicate ideas and questions either anonymously or as named communications depending on stakes the instructor wishes to place on the communication.

Additionally, tools like clicker devices are often incorporated into the technology of a SCALE-UP room to provide an anonymous to the students, but known to the instructor type feedback. This also presents a buy-in opportunity for Computer Science students as creating software to mimic clicker capabilities on a desktop/laptop/mobile device is an interesting programming challenge. Thus, students would be able to both develop and use the software they are assigned to create which improves student motivation [1]. This technique of student generated software has the added benefit of providing the opportunity to spend time on code maintenance and usability analysis, two important areas in the professional landscape which often are under-served by Computer Science programs [4].

A more public venue for student to class communication is via the viewing screens in the SUSIC Lab. Utilizing software, again leveraging student work when possible or open-source/off-the-shelf software initially, we can allow for the instructor to ”grab” the student generated content and display that content to the class at large. Aside from the obvious usage of such software for informal or formal presentations of work, this can also be used for class based debugging where problem code encountered by one group can be provided quickly to the entire class. This allows for leveraging of the entire-class’ hive-mind and also encourages dialog and discussion as students encounter other ways of solving problems in code that they may not have thought of individually[9].

It also possible to take advantage of an open-source course management software project and have activities where students build or modify existing the course management code on a development server. This will permit students to understand how to integrate their work with other, existing work. Depending on the quality of work, this would also provide opportunities for students to get their code either added to the department’s course management software or even the actually open-source project itself. Again, the focus is on creating a buy-in for the students with this room. By having their own software as part of the classroom itself, they are forced to take ownership of the room and, hopefully, the learning inside of it.
Beyond these more traditional SCALE-UP software demands, Computer Science instruction utilizing this style of group based co-operative learning must also include some way for students to quickly share code. A code repository system with version management should be utilized in a SUSIC Lab to ensure maximum code portability between group members as well as documentation trail for who added what code, when and why. This has an added benefit of acting as a transparent identifier of which students in the group had the highest workload and most well-developed code. This is important so students know they will be graded fairly, even if their group members slacked off and didn’t fulfill their group obligations.

4 Student/Instructor Interaction

Now that we have a SUSIC Lab, we need to be able to use it. How then, can we take the ideas and approaches from SCALE-UP and incorporate them in ways that are applicable and beneficial to Computer Science? Let us start with the crux of the argument for SCALE-UP, that students learn more (and better) when interacting with their academic community on interesting, challenging and engaging tasks[8]. If this is true, then lecture classrooms are ill-suited for instruction. If this is true, a SUSIC Lab would seem to possess significant upgrades to encourage and facilitate the required interaction. What about our existing paradigm of lectures and labs, separated? Labs are rarely in perfect lock-step sequence with the lectures, instead students often find that labs are days or even weeks behind the lecture for logistical reasons (multiple lab sections, scheduling conflicts, etc)[2]. A SUSIC Lab provides for the lab and the lecture to be ONE AND THE SAME, just as is the goal for SCALE-UP[2]. Unfortunately, simply having the environment and the tools needed for SCALE-UP style learning does not learning make.

As instructors in SCALE-UP style learning environments like the SUSIC Lab, we need to be able to take a step back from the front of the room. While we are still important cogs in the machinery, we are no longer in the spotlight. Instead, we should strive to be like the machines we program, omnipresent in the background, orchestrating grand events, while the user (student) gets the glory and achievement. In some Computer Science courses this Machiavellian behavior is easy. Advanced Software Engineering and Testing projects are easy to modify for SCALE-UP style learning. Instead of strict rules of engagement for programs, craft ill-defined projects with open-ended possibilities and obvious (and subtle) contradictions. Instead of ”The Instructor”, wear the hat (literally if you want to) of ”The User”. You don't really know what you want but you have an idea. Make the students coax that idea out of you piece by piece. As the projects progress, integration and unit testing can be done at the table level and beta testing performed by other students. Incorporating a ”bounty” system on bugs where finding bugs provides additional points to the finding group without hurting the scores of the developers who created the buggy code provides an incentive to actually test and peer review code instead of just ”rubber stamping” it as often happens in courses where the lab is a standalone course or non-existent. Having the SUSIC Lab be the primary teaching location also means that advanced program and project based courses can better fine tune their scheduling and feature delivery dates which produces a higher quality of work for equal student effort[12].

Introductory courses require a bit more planning and foresight. Since ill-defined problems might create significant confusion as students who are still learning what programming is and how to think like a programmer, they must be used sparingly and with ample consideration of possible exit strategies in-case they go horribly sideways. Instead, focusing on the basic skills and toolboxes instead of traditional repetitive labs is a more prudent first step. These can be debugging exercises where students must demonstrate what the bugs in a provided program actually are, how they identified the bugs as being bugs, how they discovered where the bugs were located, provide fixes along with explanations of how the fix works, and test-case generation to prove that the fix worked and did not induce other side effects.

Another possibility is to provide students with API's of new methods, functions or objects and letting them figure out how to use those methods correctly and in context. This often results in what I call a ”Student Generated Lab”, where I as the Instructor provide the goal of the lab (Example: Demonstrating that the student can properly use different looping styles), but the student has to determine appropriate contexts in which to use the different loop styles appropriately. This forces them to make the connection between when different loop styles are more appropriate than other others in addition to merely creating a loop to satisfy the often arbitrary demands of a loop writing exercise.

To prevent students from just trying to coast on their group members' work, you will have to incorporate SCALE-UP mainstays such as having individual group members explain their findings and solutions as well as the process by which those findings and solutions came about, or to have your groups create contracts at the beginning of the course that will govern their behavior and outline the penalties to be enacted by the group if a member fails to live up to their responsibilities[2]. This requires all group members to be accountable for being able to explain the process since if they are chosen speak for their group, as they must be able to understand the material well enough to do so, and the contracts allow group members who feel they are unfairly burdened by an absentee or disinterest group member to have a means to receive compensation for that burden in some respect. As the instructor, we would be able to apply the penalties provided in the group's contract, but by limiting ourselves to those contracts which the group created and agreed to, students have a level of buy-in to the process that results in a positive peer pressure from a group determined penalty instead of an instructor mandated penalty[2].

5 Future Work

After the final phases of the UND SUSIC Lab complete in Summer of 2013, a more focused reporting and analysis of student learning within this SCALE-UP inspired room can commence. Dissemination of that analysis will mark the opportunity to ask questions of the Computer Science academic community to assist in solving problems with the SUSIC Lab implementation of SCALE-UP as well as provide fully developed and tested course modules which were successfully deployed in the SUSIC Lab. Sufficient data will be gathered in the next year for the beginnings of an analysis of the SCALE-UP techniques use in the SUSIC Lab with respect to retention, absenteeism and student success in Computer Science compared to the positive results seen by other STEM fields[2].

References

[1] Allen, E., Cartwright, R., and Reis, C. Production programming in the classroom. *ACM Sigcse Bulletin* (2003), 89–93.

[2] Beichner, R. J., Saul, J. M., Abbott, D. S., Morse, J., Deardorff, D., Allain, R. J., Bonham, S. W., Dancy, M., and Risley, J. The student-centered activities for large enrollment undergraduate programs (scale-up) project. *Research-based reform of university physics 1*, 1 (2007), 2–39.

[3] Brooks, F. P. *The mythical man-month*, vol. 79. Addison-Wesley Reading, Mass, 1975.

[4] Carroll, J. M., and Rosson, M. B. A case library for teaching usability engineering: Design rationale, development, and classroom experience. *ACM Transactions on Computing Education / ACM Journal of Educational Resources in Computing 5* (2005).

[5] Dick, A. J., and Zarnett, B. Paired programming and personality traits. *XP2002, Italy* (2002).

[6] Gaffney, J. D., Richards, E., Kustusch, M. B., Ding, L., and Beichner, R. J. Scaling up education reform. *Journal of College Science Teaching 37*, 5 (2008), 48.

[7] Heller, P., and Hollabaugh, M. Teaching problem solving through cooperative grouping (part 2): Designing problems and structuring groups. *MAA NOTES* (1997), 207–216.

[8] Knight, R. D., and Burciaga, J. R. Five Easy Lessons: Strategies for Successful Physics Teaching. *American Journal of Physics 72* (2004).

[9] Lasserre, P. Adaptation of team-based learning on a first term programming class. *ACM Sigcse Bulletin 41* (2009), 186–190.

[10] Nagappan, N., Williams, L., Ferzli, M., Wiebe, E., Yang, K., Miller, C., and Balik, S. Improving the cs1 experience with pair programming. In *ACM SIGCSE Bulletin* (2003), vol. 35, ACM, pp. 359–362.

[11] Nosek, J. T. The case for collaborative programming. *Communications of the ACM 41*, 3 (1998), 105–108.

[12] Shaffer, C. A., and Edwards, S. H. Scheduling, Pair Programming, and Student Programming Assignment Performance.