The Integration of the Software Studio Approach into the Undergraduate Computer Science Curriculum

J. Ben Schafer
Department of Computer Science
University of Northern Iowa
Cedar Falls, IA 50614-0507
schafer@uni.edu

Abstract

The computer science professional solves problems for a living. When confronted with a new challenge, the professional must use the technical knowledge that is common to all solutions. At the same time, each problem presents new difficulties that demand new solutions. The goal of a computer science curriculum should be to help students learn how to approach problems requiring both solutions. While many "traditional" courses teach students to solve problems involving the former, they often neglect the challenge of teaching the latter. One approach to this challenge is through the integration of studio-based activities into courses. This paper will present the author's experiences employing a variety of studio-based techniques into different undergraduate computer science courses. It is designed to show the application of the studio approach at differing levels of "commitment," and create a dialog between educators interested in considering this approach.

1 Introduction

> *Good design and programming is not learned by generalities, but by seeing how significant programs can be made clean, easy to read, easy to maintain and modify, human-engineered, efficient, reliable, and secure, by the application of good design and programming practices. Careful study and imitation of good designs and programs significantly improves development skills.* -- Kernighan and Plauger [1]

The computer science professional solves problems for a living. When confronted with a new challenge, the professional must use the technical knowledge that is common to all solutions. At the same time, each problem presents new difficulties that demand new solutions. For many, this contrast between the analytic and the artistic is what draws them to the profession. The goal of a computer science curriculum should be to help students learn how to innovate and invent. Innovation occurs in adapting old solutions to new problems. Invention is required where past solutions are insufficient.

More traditional courses can teach the technical, analytic skills needed by a professional, but without an application to real problems these skills become purely "in head" skills and are not a true part of the student's expertise. Arguably, more traditional courses are almost entirely inappropriate for helping students to develop the creative, artistic side of their skill set. So what techniques will allow computer science educators to provide an environment where students can move these technical skills from "in head" skills to "applied" skills and allow them to explore the creative skills necessary to be a reflective practitioner? The answer may lie in the implementation of studio courses within the computer science curriculum.

1.1 What Are Studio Courses?

Studio courses have been a common teaching technique in schools of architecture and art for decades, if not centuries. Reimer and Douglas [2] describe these courses in a traditional school of architecture.

> "Each term, in addition to regular lecture courses, architecture students are required to enroll in a studio class tailored to their skill level in the program. Studio classes are held 3 days a week, 4 hr per day. Each studio class takes a real-world architectural problem and requires students to produce a final building design through an iterative design process. Problems vary by complexity of function, complexity of environmental conditions, or other aspects depending on the skill level of the students. Each week the instructor will emphasize the design of a particular architectural aspect, such as form, site location, function, and so forth, of the overall project. Depending on the studio class, students often work in collaborative teams to produce a joint design."

A studio course is more than simply a change in the number of hours or the types of problems addressed in the course. Studio courses are typically built around material where the process, the *design* of the project, and a continual analysis of both project and process are as much a part of the evaluation of the project as the resulting project. As such, a key component of a studio is the inclusion of a period of time known as the "design crit" [2].

> "The design crit is the central means of conveying design knowledge. Instructors usually gather from 2 to 4 students together at one time. Each student either brings his or her drawings over to the common meeting area or pins them up on the wall for review. Design representations are often low- fidelity sketches to promote the general communication of ideas and to enable students to throw away bad designs. While the instructor focuses on the work of one individual at a time – taking between 20 and 30 min – the remaining students benefit from the comments made by the faculty member and student.
>
> Design crits start with the student explaining how he or she is meeting the particular design emphasis for the week. To keep the critiques positive, reviewers generally begin their comments with statements like ''I like what you've done with –''. Many reviewers then use the Socratic method to ask the student a number of strategic questions which serve to highlight perceived weaknesses with the design. Reviewers often end their critique by suggesting similar problems/solutions done by well-known architects, and by asking the student if he or she has any specific problems and/or questions they wish to ask. Finally, faculty reviewers will also make helpful suggestions on the student's presentation itself (e.g., urging the student to frame the problem and to discuss his or her goals overall before getting into details). This provides the student with direction for future success, both in the current class and elsewhere. "

1.2 Studio Courses in Computer Science?

While the studio approach has been in existence for years, it has only recently begun to be applied to the field of computer science – particularly in the domain of software development. Several companies such as RoleModel Software [3] have begun to employ a software studio approach toward the development of their products

In a traditional design studio in a field such as architecture, the studio is designed to complement one or more classroom based courses. As such, the studio is designed to provide students with an opportunity to apply and discuss concepts learned initially elsewhere. While there are parallels between this structure and the lecture/closed-lab structure common in many computer science departments, there are several fundamental differences. The largest of these is what actually happens in the traditional CS "closed lab" vs. what happens in studio. In a closed lab situation students are often involved in

relatively “cookbook” activities under instructor/TA supervision. Most of these activities have a goal of simple completion, and little time is spent discussing *how* and *why* the code is written the way that it is written.

Studio courses indicate a different course organization and reinforce the intent that a course pays special attention to the artistic skills of the profession. Such courses require a design where students are put into a constant state of questioning. The student is forced to explain and defend the choice of proposed methods, processes, solutions, and implementations. She must relate these choices to other parts of the problem and solution and convince others of their adequacy. The course "instructor" metamorphoses into something of a coach: a sounding board for ideas, a constant critic who helps the student see other alternatives, and a source of direct instruction when new technical knowledge is needed.

2 Different Ways to Incorporate the Studio Approach, or “Fifty Ways to Lead the Others”

Often, at first glance it sounds as though classroom organization is particularly easy when using the studio approach. After all, it would appear there are fewer lectures to deliver, seemingly less class preparation, etc. Upon further reflection, it turns out that implementing the studio approach into existing classes can be relatively difficult and time consuming. As such, instructors interested in implementing such an approach into their courses must consider how much time they are able/willing to put into the development of such courses. The following section will discuss three different approaches with significantly varying degrees of commitment for integrating studio-like activities into undergraduate computer science courses. These include the use of in-person grading, the implementation of a “weekly” studio into a lecture based course, and a largely full-scale studio course.

2.1 A “One-on-One” Studio

While the design studio approach may interest many instructors, it is a significant change to the way most are used to running their classroom. One technique that can provide a fairly “low investment” introduction to the overall *process* of the design studio is the incorporation of “in-person grading” [4], [5]. This technique has been successfully used at the University of Northern Iowa in classes ranging from Introduction to Computer Science in Java [5] and Object-Oriented Design and Patterns (CS I and II respectively) to COBOL and Algorithms.

Instructors who choose to use in-person grading sessions require their students to schedule one or more personal and private meetings over the course of the semester. There are several different structures for how frequently such sessions occur depending on the instructor and the course topic. In some cases, all students are required to participate in several in person grading sessions for the same set of pre-arranged

assignments. In other cases, students are divided into sub-groups and rotate on an assignment-by-assignment basis. The advantage of the former is that the assignments can be selected so the instructor is meeting with students to discuss the most interesting assignments or the ones with the most flexibility in design. The advantage of the latter is that the weekly load for the instructor is fairly consistent – a small amount of time frequently rather than large amounts of time infrequently.

Prior to an in-person grading session the instructor considers the student's assignment (normally code) for "correctness" and to preview which key concepts to discuss with the student. During a 20 to 30 minute in-person grading session, the instructor and the student discuss the student's solution and the decisions the student made while completing the assignment. For example, a typical in-person grading dialogue might consist of the following:

> How did this assignment go for you?
> Walk me through your code?
> Show me the code executing.
> Why did you choose to …?
> What would your code do if …?
> What if we wanted to …?
> What would/could you do differently?
> What did you learn?

While this technique *may* be used purely as an evaluation technique – as a way for instructors to obtain a more accurate picture of the student's understanding of the material – it may also be used as an opportunity for the instructor to teach – to discuss why decisions were made, what tradeoffs were consciously considered by the student (vs. those which were merely artifacts), and to discuss with the student alternative solutions which may be better, or in some cases, worse, and why. Rather than simply having students *produce* solutions, in-person grading provides students with the opportunity to begin to *evaluate* and *critique* solutions. In-person grading has been particularly helpful as a teaching tool in courses where design of the solution is a fundamental part of the goals of the class (e.g. OO Design and Algorithms).

While in-person grading is a technique that is relatively easy to apply, it may cause instructors to experience a fair amount of déjà vu. The problem with meeting with individual students in a one-on-one situation is that instructors may spend a significant amount of time leading different students to independently reach the same conclusions. While this may be of higher benefit to the individual students, it can create significant time constraints on the part of the instructor. Similarly, in-person grading does not allow an entire class to benefit from the wisdom/experiences of a single individual without a significant effort on the instructor's part to recreate a given scenario from an in-person grading session during large-group time.

2.2 A Part-Time Studio

One of the real challenges of using a studio in teaching is that studio time is supposed to be about application/evaluation of knowledge rather than the acquisition of knowledge. For example, recall that studios in schools of art and architecture are designed to allow students to apply concepts learned either in previous semesters or to apply material learned elsewhere during the current semester. How are such activities scheduled in computer science; in particular when the studio is designed to complement classroom material learned in the same semester in which the student is completing the studio? One technique for integrating a studio-based approach into the classroom is the creation of a "part-time" studio. The following explanation of the part-time studio will be framed in the context of the "User Interface Design, Implementation, and Evaluation" course (UIDIE) being currently taught at the University of Northern Iowa [6].

The UIDIE course at UNI is a team-project based course in which students complete a large-scale software project over the course of the semester. The UIDIE course is offered within a "traditional" scheduling block. That is, it meets for 50 minutes three days a week. In a typical week Monday and Wednesday are spent in "knowledge acquisition" activities. That is, students are either listening to lectures or are participating in active learning activities. For example, a week's topic might be how to gather information from users to complete a task-analysis, the generation of paper prototypes, the completion of a cognitive walkthrough, the completion of a heuristic evaluation, and so on. By the end of Wednesday's lecture, students have gained enough knowledge to begin the next deliverable (typically due the Friday of the *following* week) for their semester long project.

Fridays are spent "in studio" performing design crits on the week's deliverable(s). During the design crits, teams take turns presenting and discussing their deliverables. Early in the semester these take the form of short presentations by each team followed by a series of, frequently instructor led, discussion generation questions similar to those used in in-person grading. Depending on the deliverable, students are expected to defend the "what," "why," and "how" they went through in generating their deliverable. Furthermore, questions may be directed towards the "other teams" to question alternative approaches or to identify problems unseen by the "on the spot" team.

As the semester progresses, the format of these design crits tends to shift. At first, as students begin to become familiar with the process, members of the "other" teams start to lead the discussion after team presentations. Remarkably, this seems to occur with little explicit encouragement from the instructor. This quickly "evolves" into a less structured presentation/question format and the process begins to take on the feel of a truly open studio. Teams make their presentations but are constantly interrupted by other teams to ask questions, challenge assumptions, and suggest alternatives.

The "part time" studio approach allows instructors to structure their course in such a way that new content can be interspersed with in-depth participation with, and analysis of, this content. Furthermore, the advantage that this studio approach has over simply having

students participate on a team is that the studio gives them exposure to the process for more than one team. A team who struggles with a deliverable can gain insight by observing a team that has done it well. Even when all teams have done it well, each project provides a different approach to the design and implementation of a usable product. Furthermore, the inclusion of the studio puts an additional emphasis on the fact that the knowledge to be learned in the course is more than simply some facts thrown out by an instructor. As the semester progresses, the studio allows the students begin to see that the material presented is all about the process and the creativity learned by *participating* in the process.

2.3 A Full-Time Studio

While the part-time studio works well as a complement to the application of material currently being learned, many computer science curriculums also have a limited number of situations where students learn material in one semester and truly start to apply it in subsequent semesters. When this is the case, follow up courses may actually be structured as "full-time" studios. The following explanation of the full-time studio will be framed in the context of the "Intelligent Systems" course taught at the University of Northern Iowa [7], [8].

Similar to the UIDIE course, Intelligent Systems (IS) is a semester long, team-based project course. Unlike UIDIE, Intelligent Systems has a prerequisite specifically related to the content of the course. Students must have previously completed the department's Artificial Intelligence course. At the very start of the semester, students divide into teams and identify a project using one or more techniques they learned in AI. From the very beginning, activities are tailored towards the type of projects selected by the students.

The day-to-day structure of the IS course is much less rigid than that in a traditional course. While students are required to meet on a regular course schedule (the course has been taught using both a three-day a week and a two-day a week format) what is done during those regular meetings is highly flexible, and modeled loosely after the studios in art and architecture. On any given day an observer might discover students performing one of three activities; presentation/discussion of "content," project design crits, and open lab workdays.

On content days (frequently Mondays), the entire class has previously read a reading related to the design of intelligent systems. These may come from a standard required text, or they may come from the research literature. Ideally, the readings are selected for their applicability to the types of projects being completed during the given semester. A team of two students (not necessarily working together on the same project) leads a whole class discussion of the specific content of the readings. This if followed by a discussion regarding how this material is related to each team's project. Some weeks, the topic is a perfect fit for a particular team's project (an in depth reading on a variety of learning techniques in neural environments in semesters when students are building neural networks). Other weeks, the topic is less ideal (the discussion of an article on

MYCIN for the same team). However, students are encouraged to consider how they would make a technique work in their project ("Suppose you are offered BIG bucks to build a system that does X using technique Y. How would you make it work?"). Finally, they are asked to defend why a technique is (in)appropriate for their project. In doing so, students begin to spend less time thinking about *how* a technique works, and more time with analyzing *where and why* it might work.

Design crit days (typically Wednesdays) are very similar to those described with the UIDIE course. Design crits in the IS course involve a single team discussing the current status of their project. Initially these sessions involve students explaining their project to the other teams and identifying the scope of the project they hope to complete and the techniques they are considering using. As the semester goes on, these begin to become much more detailed as teams explain tools/techniques they have discovered, how they are evolving their project, and how the project continues to fit the definition of an "intelligent system." While this may sound very similar to the project updates used in many other project based courses, the fundamental difference lies in the interaction between the "on the spot" team and the remainder of the class. Students from all teams are encouraged to become actively engaged in an analysis of the decisions made by both their team and the other teams in the class.

Open work days (often Fridays) are much like "work days" in non-studio based courses. Students are given in class time to meet with their teams, conduct research, write code, and ask for assistance from the instructor and fellow classmates. In keeping with the spirit of the studio, students are encouraged to interact with each other and members of the other teams. In "traditional" classes, students are encouraged to "work alone." In studio, students are encouraged to take advantage of the experiences and expertise of their fellow classmates. As students look for tools/algorithms appropriate for their project, they often stumble across things appropriate for other groups and are encouraged to share. If the instructor is aware of a particularly interesting discussion occurring, she may bring the other groups over to include them in the process.

In all three of these activities, the fundamental difference between the studio approach and the non-studio equivalent is the amount of interaction students have with each other and in the amount of time spent in the role of critic or analyst. Students rapidly learn to become an active participant in presentations and projects of other students. They quickly discover that thinking about problems in other people's projects can lead to insight into solving problems in their project.

3 Discussion

While the prior section might lead one to believe that implementing a studio approach within a CS class is a simple matter, that conclusion would be erroneous. In fact, the process can be difficult, time consuming, and a challenge for both students and instructors who must modify how courses and class times are structured.

One of the most difficult adjustments for an instructor to make is to overcome the initial feeling that the course has no structure. This is often the case because the instructor fails to plan appropriately for the studio (although it can also be the case even with well planned studios). On first glance, it seems like all an instructor needs to do to conduct the studio is to walk in and lead a discussion. More often than not, when this approach is attempted, the instructor will find that the result is little more than a team status report or a discussion of some minor side issue to the course. Experience suggests the most successful studios have occurred when there are explicit goals in mind for the studio. It is important for the instructor to have a firm idea regarding what issues he wants the students to come away with and what topics might come out of discussions as meaningful spin-offs. This is not only difficult, but also time-consuming. In essence, the instructor needs to consider multiple game plans for any given class period and be accepting of the fact that, at best, only one of the game plans may go into play.

Having said all of this, both instructors and students alike need to be willing to make the adjustment to a course structure that allows open-ended discussions to be initiated and maintained. The instructor has to come to class prepared for the unexpected. There is a plan, but that plan may deviate if not completely disappear at any point during the day. Some days, students come ill prepared and/or unwilling to participate. On these days, the instructor must be able to ask questions, pose dilemmas, challenge assumptions, and introduce modifications that completely change the nature of the problem currently on the table. On other days, students will play this role. On yet other days, students will pose issues that were never on the instructor's radar. This can lead to days (and sometimes weeks) where the course goes a direction that the instructor had never anticipated. It requires the instructor to reconsider what is important and what topics need to be readdressed later. However, these spin-off discussions often develop because there are unanticipated issues that several students/teams are facing. In these cases, the instructor needs to be ready to let new discussions occur while continually monitoring the direction of the discussions to see if they are continuing to cause "learning" on the part of the students. For example, there is constructive complaining about lack of appropriate tools to solve a particular problem and there is just simple complaining. The former allows students to consider why such tools do not exist or may be difficult to maintain while the later simply allows students to waste time complaining.

Despite these difficulties, both anecdotal and experimental results suggest that the end results are well worth the efforts. In the IS course one student was recently discussing his project to train a neural network to perform OCR with computer-generated fonts. The images he was using were simple graphics files 18x24. He was commenting that while that seemed small, the 432 inputs necessary for the network was causing a minor challenge. Another member of the class observed that the image files all seemed to have some white-space padding around the actual characters and questioned whether the files could be trimmed by finding rows/columns which were nothing but white-space for all character files. A third student, who had been spending a fair amount of time working with bitwise operations for his team's project, immediately suggested converting the image files to a bit String (something likely to happen anyways in order to more efficiently process the training data) and performing a cumulative OR over the set of

data. Any bits still set to 0 would be inputs unutilized in the network (although the network should LEARN this fact). A fourth student proposed that performing a cumulative AND over the set of data would indicate any bits which are always on. In hearing this, the third student suddenly realized one of the solutions to a problem he had been having with his project involved a similar solution (actually using XOR). The initial student learned a VERY elegant technique that might help him reduce his problem slightly. The third student got to be an expert and help out, but also learned a possible solution for his own problems. The whole class benefited from the exchange by seeing a completely "different" way to solve the problem.

In another example from the IS course, in a recent design crit, one team was discussing how they have decided to write some wrapper code so that they could get two existing tools (a natural language processing tool and a knowledge management tool) to work together within the context of their project (a system to read story problems and write out the algebraic equation which will solve the problem). A non-team member made the comment that he couldn't imagine going to all the trouble to make existing, generalized, public domain tools work in a specific application. He would rather start from scratch and simply build the tool to do what he needed it to do. This led to an extremely long discussion (that continued to a second class) about the tradeoffs between using existing tools vs. writing your own tools and under what circumstances each seemed to be the most appropriate approach. Students learn the most from these discussions when they come up under their own power rather than when an instructor decrees, "Today we want to discuss X." Students could immediately relate to the question and find a position to fight for, yet by discussing it in the group context of multiple team projects, begin to consider why their initial answer might not be the one, true "right answer." This type of discussion came about because of the class structure established through the use of a studio approach to learning.

While little has been done to experimentally to consider the affect/effect of full or part time studio courses in computer science, there has been initial work done to consider the affect of in-person grading [5]. Initial results of a controlled user study show that there was no effect on grades or in-class participation of those students who participated in in-person grading sessions. However, the same study showed that students were much happier with their overall course experience and *felt* like they learned more (even though the data shows they did not). In a field where retention of students has become a significant discussion point, any technique that seems to improve the students' attitudes toward their coursework seems to be a technique worth considering.

All in all, these experiences have left the author feeling confident that studio courses have a place in the undergraduate computer science curriculum. In order to fully enjoy the benefits of these techniques, however, it will require the continued efforts by instructors to try different mechanisms for implementing such open ended analysis techniques into a variety of courses and sharing which techniques work and which techniques do not.

4 Acknowledgments

The author would like to thank Drs. Gene Wallingford and Philip East for their direct and indirect contributions to the author's understanding of in-person grading and the studio approach as well as for continuing to fuel the author's passion for figuring out how further application of these techniques can improve the way he teaches.

5 References

1. Kernighan, B. and Plauger, PJ. *The Elements of Programming Style*. McGraw-Hill 1974.
2. Reimer, Y.J. and Douglas, S., "Teaching HCI Design With the Studio Approach", *Computer Science Education* Vol. 13, No. 3, special issue on Human-Computer Interaction, September 2003, pp. 191-205.
3. http://www.rolemodelsoftware.com/process/whatIsXp.php
4. East J.P., "Experience with in-person grading." In Proceedings of the 34th Midwest Instruction and Computing Symposium. 2001.
5. East, J.P., and Schafer, J.B, "In-Person Grading : An Evaluative Experiment," Proceedings of the 2005 ACM Conference of the Special Interest Group, Computer Science Education (SIGCSE-05), pp. 378-382.
6. http://www.cs.uni.edu/~schafer/courses/112/
7. http://www.cs.uni.edu/~schafer/courses/162/
8. http://www.cs.uni.edu/~wallingf/teaching/162/