Rethinking the Computer Lab: an Inquiry-Based Methodology for Teaching Computational Skills

Nick Senske

University of North Carolina at Charlotte

Introduction

Two years ago, the UNC Charlotte School of Architecture began a new curriculum initiative to teach computational design to all of its students. As a capstone to the beginning design sequence, the school now teaches a required methods course to over 70 students a year. The objective of the course is for students to learn essential computational skills and ideas that will prepare them for parametric tools such as Revit and advanced digital techniques found in later studios. However, learning how to think about and make things computationally can be difficult for many students. Issues of affect and conceptual misunderstandings must be overcome in order to learn skills and knowledge that will transfer beyond the course. This paper attempts to address the question: How can schools of architecture teach computation in a manner that is engaging and successful for a large and varied cohort of students? The author proposes that inquiry-based labs, which are more active and motivating than traditional tutorial-based labs, can address some of the challenges of teaching computation to architecture students. In support of this claim, the author presents the findings of a two-year pilot study to examine the effects of the new labs.

Issues with computer labs

The "tutorial-based" lab is a common form of computing instruction in architectural education: an instructor demonstrates a particular method on a screen and the students follow along, asking questions and receiving help or feedback as they work. The benefit of the tutorial structure is that everything is organized from a script. Because the students are all following the same predetermined prompts, it is not difficult for the instructor or TA to teach the lesson, measure outcomes, and offer assistance. Thus, tutorials are a straightforward way to communicate procedural information to students.

Despite the popularity of tutorials, there are several issues with the format. The main one is that tutorial-based labs are a passive experience. There is seldom time to deviate from the lesson for discussion or exploration. Additionally, the pace of the lab is not controlled well and does not support different styles of learning. Students are dependent upon the teacher to present the lesson. Some students cannot keep up with the material; others are bored by what they see as a slow delivery. This can be a problem for the teacher, as the whole class can only progress at the speed of the slowest student. Pauses to answer questions, clarify explanations, and troubleshoot software problems delay the lesson and reduce the amount of content that can be expressed in a lab session. Pacing issues like these can make classroom management difficult for the instructor. Because all of the students have computers, maintaining attention is an ever-present concern. It is easy for them to become distracted by email, the web, or anything else that can be brought up on the screen. Students that are not paying full attention to the lesson or trying to work ahead might miss a step, which further delays the class. Students that get too far behind might start to work on their own or do something else, which defeats the purpose of the

lab entirely. Overall, the dependencies between the students and instructor are too great and represent a flaw in the tutorial-based lab method.

Generally speaking, active and engaged learning is more effective than passive learning[1]. Following along with a tutorial is better than listening to a lecture, but to facilitate deep learning, one has to put information and skills to use in other contexts[2]. This is why, in tutorial-based courses, most learning tends to happen in the assignments outside of class. A problem with this is how inefficient learning can be in the assignments that follow tutorials. Students tend to have poor recall of what they did in the lab. It is difficult to take good notes while trying to follow along in class. If notes are provided, these are often insufficient or incomplete. So the students have to try to remember the procedures from class while attempting the assignment. Frequently, the assignment is not broken down in such a way as to recall the steps. Thus, students end up struggling to recreate the steps and engage with the assignment superficially. Assignments should be where knowledge is integrated and extended, but when students feel left adrift after the tutorial, the potential of the assignment is eroded by recall and repetition.

How students learn is not the only concern. The skills students learn in tutorial-based labs tend to be inflexible, locked into rote sequences of discrete steps and disassociated from essential conceptual knowledge. This leads to design work that lacks criticality and is too narrowly focused on the parameters of the tutorial. Students learning from tutorials may not be able to abstract the important principles, which can cause problems if they try to do something else with the software or when the software inevitably becomes out of date. Because this knowledge is tied up in specific procedures, it is unlikely to help them learn other skills and programs. Lab tutorials cultivate a singular way of generating computational artifacts, and do little to teach students how to go about designing through them. A course grounded in tutorials risks being about the commands and procedures and less about learning what is important about computation

Another weakness of the lab-based tutorial is that it does not accommodate different learning styles. Some students need to hear and see the lesson. Others just want to jump in and do things. Some want to work in groups to learn together; others want to work alone. In addition, the tutorial lab has not kept up with educational research and changes in student learning styles as a whole. Research suggests that people cannot focus on a lecture for more than 18 minutes[3] at once, and so course styles are changing. Students are learning in a variety of new ways, with online materials and in smaller chunks of time. As it stands, the traditional lab is not flexible enough to accommodate the way students are said to learn best.

As described in this section, there are many reasons to reconsider tutorial-based labs for teaching computational skills and knowledge. The computer lab, as a space and an idea, will not be disappearing soon, but it does seem to be in need of revision due to changes to technology and learning habits. In light of this, the question becomes: What can be done with labs to leverage them as spaces for a different and more productive kind of learning?

The context of the class at UNCC

Before describing how the school revised the computer labs for its computation course, some context for the changes may be helpful. Computational Methods is a 15 week course and a requirement in the UNCC architecture program. Because of this, the course has an enrollment of over 50 undergraduates and 20 graduates a year. Course topics include basic computational concepts such as variables, iteration, and data structures, but with an emphasis on connecting these ideas to design principles and precedents. The goal of the course is for students to achieve an awareness of

computation and basic skills for representing and designing via computational processes. The course is not about learning computation, but rather learning about design via computation.

The course emphasizes thinking about computation in a software-agnostic manner, but practically speaking, lessons primarily use the Grasshopper scripting language[4]. Grasshopper was chosen because, at this point in the curriculum, UNCC students already know Rhino. Based upon his research, the author felt it was best to build upon students' preexisting knowledge. Also, because Grasshopper is a visual scripting language, it reduces many of the syntactical problems with programming that can challenge novices. Although Grasshopper does not explicitly connect with all aspects of computation (e.g. iteration and recursion are handled implicitly), nevertheless, it serves as a motivating introduction.

In the curriculum, students meet two times a week, first in labs of fewer than 20 students and later in a lecture with the entire class. Students submit weekly assignments and complete a midterm and final project of longer duration. Other papers by the author contain more information regarding the curriculum[5] and syllabus[6].

Inquiry-based lab design

The typical computer lab bears little similarity to the conventional idea of a laboratory. A science lab is an active place where discoveries are made and practical and abstract knowledge come together. The tutorial-based computer lab is a passive place where knowledge is consumed and enacted, without much for the student to discover. Because of this, the potential for deep learning is low. But what if the computer lab were more like a science lab? This is the thesis of the current version of Computational Methods. The course is built around the idea of an inquiry-based lab, where the emphasis is on constructing meaning through experimentation. By proposing and answering questions, students learn more than software commands; they learn how to think like computational designers.

To begin, what does "inquiry-based" mean and where does the term come from? Inquiry-based learning can be traced to the medical and nursing professions, which changed from a lecture-based learning model to one where students are taught by solving problems in realistic situations[7]. It is closely related to project- or problem-based learning, but a key difference is that, with inquiry-based learning, the teacher provides information directly to the students to facilitate the learning process. As one might expect from the name, the goal of inquiry-based learning is to seek out and integrate knowledge, rather than merely being exposed to it.

Online media and the inverted curriculum

The labs are essential to the course but are not the only component within it. In advance of the lab, students first watch online videos demonstrating the skills they need, which they follow along at their own pace. Students do not require instructor interventions to study command knowledge and procedures, which eliminates many of the problems of pacing and classroom management described earlier. Online media also frees up class time to explore the meaning and implications of these elements. This arrangement is an example of what is known as an "inverted curriculum"[8]: the students do things outside of class that are usually done in class, such as learning command knowledge, and in the labs, the students work on assignments, which they would typically do outside of class.

Lab reports

The primary assignment type in Computational Methods is a "lab report," a series of in-class prompts given to students to facilitate inquiry. In a typical computing course, the assignments tend to involve making something using the tutorial elements. This has two potential problems: the work can become too constrained by the tutorial or the student does not know how to

extend the tutorial to produce new work. In both instances, the assignment emphasizes production rather than comprehension, which is counterproductive for deep learning. In the author's experience, learning scripting while attempting to design with it at the same time can be too demanding for many students. Thus, instead of asking students to make artifacts prematurely, the lab report is designed to promote and assess comprehension. In the report, students ask what the pieces of a script are doing, reflect upon how the algorithm functions within a design, and discover for themselves how to make the software do things that were not previously demonstrated in a tutorial. This is the core idea of the inquiry-based lab: students are not this told information in a decontextualized way, as happens in a tutorial. Instead, they build upon what they know and find it for themselves.

Most lab reports follow the same basic structure. They begin by asking students to practice specific techniques from the video. Practicing is important because it builds confidence and helps the students recall the commands and steps they need for the report. In later prompts, changes are added to the script to produce something unexpected by the student. This is motivating and provokes the questions to follow. The next series of prompts ask students to conceptualize and explain what is happening in the new script. Last, students modify the script to make some variations of its initial state. Students write their explanations, solutions, and reflections and submit the reports to the instructor. Explication of process is an essential practice for making sense of computational artifacts. In addition, it provides a means for the instructor to assess students' thinking and address any misconceptions. In the inquiry-based lab, student discovery and understanding is valued above good-looking designs of dubious comprehension.

Design in the lab

For novices, the lab sequence is better than a design assignment because it is highly scaffolded. The steps and expected outcomes are clear and do not ask students to go too far beyond what they already know. Scaffolding is more motivating because students have a higher chance of success with incremental steps rather than being left to their own devices. This is not to say that there is no design in the course, only that it is carefully regulated. For example, many of the later prompts in the lab report leave room for students to expand upon the solution. All of the students must demonstrate their mastery of a principle, but they also have the opportunity to express themselves and to make the submission as complex as they want. Students that are motivated appreciate the flexibility and those who are having trouble keeping up can simply opt out.

The midterm and final projects for the course are more independent and allow for personal exploration of computation. The midterm is somewhat scaffolded study of a precedent building. Students extract a parametric module and then repurpose it to create a pavilion on a provided site. The final project is an iterative study of a building element taken from a past or current studio project. Both of these projects are well-constrained with a clear path to follow. They model the computational design process while allowing the student enough freedom to make the process their own.

Social scripting

During the lab sessions, two students work on a script at the same time. In agile software development, this method is known as "pair programming"[9]. One student writes the script and another student observes and offers feedback. Students trade roles frequently, so they experience both ways of working. Pair programming is helpful because it reduces the cognitive load on the students, which can improve performance[10]. When students first learn scripting, it can be difficult to know how to assemble the program and to understand what the program is doing at the same time. This way, the labor is divided and students can work

together to study the question and explain their solution.

A side effect of pair programming is that it makes learning computation more social. Students who might otherwise feel that they could not write scripts are encouraged when they can work together with someone else. They soon find that programming is not as intimidating as they thought. In addition, the students felt free to ask questions of other groups. One might expect that this led to plagiarism or students otherwise coasting through the report, but this did not seem to be the case. Most of the time, students did not want an answer, but rather a discussion about their explanation. Because the goal of the lab is comprehension, the stakes are purposefully low. Students that finish their work and show effort are graded well. An incorrect explanation is not penalized; it is remediated. Students learn quickly that understanding the previous lab is prerequisite for the next one. Lab partners expect of each other to keep up with the videos and reports. By creating a culture of experimentation, the lab becomes intrinsically motivating and cheating is disincentivized.

Methodology

To study the effectiveness of the new lab format, the author collected data from a control group and experimental group and compared the two. In fall 2011, the control class was taught with a standard lab format. An instructor gave tutorials to students, who followed along at the computer. The following year, an experimental group was taught using inquiry-based labs. Before and after both courses, the students were given a Likert-scale survey in an anonymous online format. All surveys were voluntary. For the purposes of reporting results, the Likert scale was combined so that, for example, Strongly Agree and Agree were counted as agreement with a statement. The instructors also collected written feedback from the university's online student evaluations.

The response rates for both surveys were high, averaging 68% of the class for the Fall 2011 course and 74% for the Fall 2012 version.

Results

The study revealed benefits from the inquiry-lab design over our former tutorial-based labs. Most significantly, student affect was improved by the inquiry-based labs. In the earlier version of the course, 36% of students claimed that they did not see the connection between scripting and their work in studio. In the written comments, many specifically said that they would rather learn BIM instead. Perhaps the most discouraging finding was that 74% did not believe that the course was important enough to be required.

Following the changes in the curriculum in the 2012 course, student attitudes improved. 92% in the second group of students reported that they found the material relevant to their future career, an improvement of 28%. Additionally, far more students – 98% of the class, compared to 58% previously – believed that learning about computation would help them learn other software such as BIM. Overall student satisfaction also improved, from 64% in the 2011 course to 90% in 2012. 81% said the course should remain required. The reception for the course was much improved with the modifications to the curriculum.

The scope of the course did not change between semesters; merely the way content was framed and delivered. Students were not more satisfied because the course was easier, but perhaps because it was a better learning experience. The inquiry-based labs gave students an opportunity to practice their scripting skills in a manner that guided them towards a better understanding of computation. The lab gave the students structure, which they did not get from the assignments in the previous class. Another key change was that, by making the lab more social, students were encouraged to share when they did not understand something. They did not worry that their peers might understand

something that they did not. Because students worked together and had coaching present in the room, many misconceptions could be overcome in the lab rather than outside of class. This saved time and reduced student frustration.

The students approved of the inquiry-based labs, but did the labs make them better computational designers? This is a fair question. It is difficult to measure the effect inquiry-based labs had on a student's design skills, and whether these skills develop better with inquiry based labs compared to other methods. This is something the author is presently studying, but it will take more testing to make this determination. While there may be some benefits to learning programming, no study – including this one – has definitively found any. So it seems unwise at this point to claim that it helps students think or design better.

What the study does say is that the inquiry students reported greater confidence in their knowledge of the material and were more interested in applying it to their designs than their counterparts in the tutorial lab. Results like this are significant because, according to the surveys, many students who take the required course are not initially interested in computation and/or do not believe they are capable of learning it well. Students who are not motivated to study – or to continue studying after a course – are unlikely to master a subject, and so engagement is a critical factor in developing skills and understanding[11]. This seems to be the primary contribution of inquiry-based methods to the computer lab.

Conclusion

The goal of a first course in any subject is not to make students a master of that subject. This is impossible. Rather, the goal of a first course is to make students aware of a subject and to inspire them to want to learn more. By reorganizing the curriculum around active, social, inquiry-based labs, the new version of Computational Methods successfully introduced a diverse group of students to a challenging subject while maintaining their interest. While the study could not prove whether students understood the material better because of the different lab, if students find the work relevant and are encouraged to experiment and continue learning, then it is likely they will perform better. Thus, inquiry-based labs are a promising method for teaching computation in architectural education.

Notes

[1] Ambrose, S. A., M. W. Bridges, et al. (2010). How learning works: Seven research-based principles for smart teaching, Jossey-Bass.

[2] Bransford, J. D., A. L. Brown, et al. (2000). How people learn, National Academy Press Washington, DC.

[3] Johnstone, A. H., & Percival, F. (1976). Attention Breaks in Lectures. Education in chemistry, 13(2), 49-50.

[4] Rutin, D. (2012). Grasshopper (version 0.8.0052) [software].

[5] Senske, N. (2011). A Curriculum for Integrating Computational Thinking. In Parametricism: ACADIA Regional Conference Proceedings. Lincoln, NE.

[6] Senske, N. (2013). Building a Computational Culture: a pedagogical study of a computer programming requirement in architectural education. In The Visibility of Research, ARCC National Conference Proceedings. Charlotte, NC.

[7] Barron, B., & Darling-Hammond, L. (2008). Teaching for meaningful learning: A review of research on inquiry-based and cooperative learning. Powerful learning: What we know about teaching for understanding, 11-70.

[8] Pedroni, M., & Meyer, B. (2006, March). The inverted curriculum in practice. In *ACM SIGCSE Bulletin* (Vol. 38, No. 1, pp. 481-485). ACM.

[9] Williams, L. A., & Kessler, R. R. (2001). Experiments with industry's "pair-programming" model in the computer science classroom. *Computer Science Education*, *11*(1), 7-20.

[10] Sweller, J. (1988). Cognitive load during problem solving: Effects on learning. *Cognitive science*, *12*(2), 257-285.

[11] Pugh, K.J., and D.A. Bergin. (2006). Motivational influences on transfer. *Educational Psychologist* 41 (3):147-160.