Turning the Tables: Learning from Students about Teaching CS1
Amber Settle
DePaul University
243 S. Wabash Avenue
Chicago, IL 60604
(312) 362-5324
asettle@cdm.depaul.edu

ABSTRACT
Programming has a central role in the computing curriculum, and introductory programming classes have been extensively studied in the computer science education literature. However, most of the studies focus on the effectiveness of various pedagogical approaches on student learning and engagement, and relative little attention is paid to faculty development. The gap in the literature puts CS1 faculty interested in effectively implementing innovative pedagogical approaches in a difficult situation. This article argues that taking a behaviorist approach to the CS1 classroom can provide much-needed feedback. Students provide instructors with one of the best sources of information about effective programming instruction, both with respect to pedagogical approaches and with respect to less formal issues such as classroom management, student-faculty interactions, and course policies. Faculty who choose to listen and learn from the comments made by their CS1 students will find a wealth of information to guide them in their development as instructors.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education

General Terms
Human Factors, Languages

Keywords
Faculty development, CS1, programming, students, pedagogy, behaviorism

1. INTRODUCTION
One of the most fundamental topics in the computer science curriculum is programming, and it is often the case that the first experience students have with computing is a programming course. There have been some recent efforts to expand the notion of computing beyond programming, including a push to include computational thinking [28] into undergraduate and K-12 curricula. Some authors who hold the position that computer science is more than programming believe that equating programming and computer science narrows the field and drives away students [11]. Others argue that programming is a more advanced skill, akin to proof construction in mathematics or literary analysis in English, and that foundational ideas and terminology need to be conveyed to students first [19]. But it is clear that programming remains a crucial part of computer science. It was identified as an essential skill that must be mastered by anyone studying computer science [10] and remains crucial in the general and tier-1 courses in the Computing Curriculum 2013 strawman document [9]. The information technology discipline also holds programming as fundamental, making it one of the pillars of the model curriculum [15]. An article discussing the development of the curriculum notes that “as with any computing professional, the IT graduate must develop the skill to program” [12, p. 354]. Courses in which students learn to program are not in danger of disappearing from the computing curriculum.

Because of their important role in computing, introductory programming classes have been extensively studied. Nearly every computing education conference contains at least one paper, panel, or poster in which the topic is a programming course, tool, or pedagogical approach, with work about CS1 courses appearing most commonly. For a variety of reasons these articles typically focus on student learning, on student engagement, or on both. The biggest reason that introductory programming courses are so widely studied is that, as stated before, students are most likely to have their first experience with computer science be a programming course and that the experience in the first course often has a strong impact on students’ persistence in the major. For example, one study found that when students perceive the pace and workload in introductory classes to be too heavy, especially in relation to their experience level, they are unlikely to pursue the computer science major [3]. For a field that experiences periodic and dramatic enrollment fluctuations [17], understanding the best way to teach students in their first class is important and worthy of deep study. Many of the pedagogical approaches to teaching programming, including collaborative and collaborative learning, active learning, peer instruction, peer assessment, studio-based instruction, and team-based learning, provide useful tools for instructors. These approaches and selected findings about them are summarized in the next section.

However, significantly less attention is paid to faculty development with respect to teaching CS1 courses. Faculty attending computing education conferences certainly have many opportunities to learn about the latest pedagogical approaches and the effectiveness of such approaches. Further, the instructor’s role in the CS1 classroom is discussed in nearly every paper focused...
on a particular pedagogical approach, and some studies have considered the impact of the instructor in the CS1 classroom [22]. But articles about faculty development for the CS1 classroom are difficult to find, and this phenomenon is not limited to the computing discipline. In her work, Adams noted “most teachers are not taught how to teach and most universities do not expect other than time-honored teaching strategies from their faculty members” [1, p. 3]. Often the only regular feedback that CS1 instructors receive about their teaching is information from student course evaluations [1]. While student teaching evaluations have been widely studied in many disciplines, the focus of this work is typically on the correlation between student learning and course evaluations [8]. Other problems with student course evaluations are the necessity of delivering the information after the conclusion of the course, which does not allow for adaptive change, and the issue that “many course evaluation instruments are poorly designed or carelessly administered” [1, p. 3].

A wealth of information on pedagogical approaches to CS1 courses may have the ironic effect of putting instructors in a difficult position. The most motivated of them may choose to adopt a novel pedagogical approach, but implementation and evaluation of the effectiveness of changes can be challenging and time-consuming. However, a look at faculty development in other disciplines provides a fruitful approach [1]. The thesis of this article is that it is productiv to take a behaviorist approach to faculty development in CS1 courses. Behaviorism “assumes that the classroom environment is rich in a series of reciprocal cues, actions, and reactions between the teacher and learner” [1, p. 6]. By carefully observing the clues that CS1 students provide, faculty can learn a great deal about effective implementation of novel pedagogical approaches to teaching programming.

In the remainder of this paper we show how pedagogical approaches that are widely studied in the computing education literature can be reinforced and augmented using information gleaned from student comments. Students in the CS1 classroom provide instructors with one of the best sources of information about the effective delivery of programming instruction, resulting in approaches that mirror many of those studied in the computing education literature. They also provide information about less formal approaches, informing instructors about productive ways of handling student-faculty interactions. The contention here is that faculty who choose to listen and learn from the comments made by their CS1 students will find a wealth of information to guide them in improving their teaching techniques.

2. EXAMINING THE CS1 LITERATURE
The thesis here is that careful analysis of student comments provides insight echoing that which can be found in the literature, which necessitates a review some of the previous work on CS1 pedagogy. Note, however, that this section should not be considered an exhaustive review of the CS1 literature. Instead, it focuses on pedagogical approaches that have been developed or modified, and in some cases widely studied, in the past 5-10 years. The intention is to provide an overview of recent trends in CS1 education. The approaches to student interaction found in the literature discussed are then summarized to find common ideas and themes.

2.1 Pedagogical Techniques in CS1
One of the most widely studied techniques employed in the CS1 classroom is pair programming. In pair programming two people work side-by-side to develop code. One is the driver who is responsible for designing and typing the code. The other is the navigator who has responsibility for observing the driver in order to detect errors and offer ideas in solving a problem. The pair switches roles at regular intervals [25]. Compared with students who work alone, students who use pair programming in introductory courses are more confident and more likely to remain in a computing-related major [23]. Pair programming results in significant improvement in individual programming skill, particularly for students with lower SAT scores [6]. Multiple studies have also found that students who use pair programming in a CS1 course are more likely to complete the course [6, 23]. In a systematic literature review of seventy-four papers, fourteen compatibility factors that can impact the effectiveness of pair programming as a pedagogical technique were identified [25]. The review found that students’ satisfaction was higher using pair programming than when working solo, and that it was effective in improving students’ grades on assignments [25]. Lacking from the literature was studies on pair compatibility factors [25]. The importance of pair compatibility is echoed in a study of first-year students’ impressions of pair programming [27]. The authors conducted semi-structured interviews with students who had used pair programming in their CS1 course and then moved to solo programming in CS2. Students reported that solo programming was more frustrating and pair programming was a good way to learn to program and prepared them well for solo programming, although scheduling is an issue in a course using pair programming [27].

A CS1 technique adopted from science pedagogy is peer instruction [26]. In this approach students answer multiple choice questions typically using hand-held remote devices (clickers), discuss the questions with their peers, and then answer the questions again. Peer instruction has been widely used in physics education for over 20 years and has been shown to be effective in improving performance on conceptual questions, but only recently have researchers considered how the technique can be adapted for the CS1 classroom [26]. One interesting suggestion to come out of the work is: “It is important to inform students about your choice of pedagogy, and explain to them the benefits of using it.” [26, p. 345]. A particularly interesting article is one that arose out of an experiment to replicate a study in biology that found students who discussed questions with their peers were better able to answer a second, conceptually-related question [24]. It should be noted that this work was done in the context of upper-level computing courses.

Another approach to facilitating student interactions is to allow students to generate content. Typically the format of this content is exam-style questions with corresponding solutions. The questions are typically peer-reviewed and are used for drill-and-practice sessions [20]. They may also be rated by students as part of the peer-review process. There are a large number of systems that have been designed to support student-generated content [21]. These systems support contributing student pedagogies as well as peer assessment [21]. It has been shown that the act of question-generation has a positive impact on performance in related exam questions, although correlations between activity and
exam performance can be explained by the level of student ability in the context of the course [20]. The authors suggest that a more sophisticated analysis of the relationship between the activities supported by the technology and the impact of the interventions be performed [20].

Another new approach in the CS1 literature is the use of studio-based instruction. In a studio-based curriculum, students are given complex and meaningful problems for which they have to construct computational solutions, and they present their solutions and justifications to the entire class for feedback. The critiques provided by their classmates are used to make appropriate modifications [14]. Proponents of a studio-based approach to the computing classroom argue that like other design sciences, metrics of success and stopping criteria for computer science projects are often unclear. This results in an iterative approach to computing projects, which lends itself to a studio-based approach [7]. Some universities have modified entire degree programs to center on a studio-based approach or applied the technique to gaming or other computing curricula [7]. One interesting adaptation of the studio-based approach is the pedagogical code review, which is modeled after the code inspection process used in the software industry [14]. In this review, students individually review code and then come together as a group to log issues with the code. The researchers on this study found no learning differences between a class using a traditional approach and a pedagogical code review approach, but they did find that student self-efficacy decreased more in the traditional course and that peer learning increased in the studio-based course [14]. An article reviewing studio-based approaches to computing curricula has noted that comparative studies between studio-based learning and other pedagogies remain sparse and that using studio-based learning as a driver for class lectures might be a productive area for exploration [7].

Many novel approaches to the CS1 classroom employ mixed pedagogical methods. There is a large body of work that suggests that active learning is more effective than lecturing, and one study found that student-generated active-learning exercises were an effective addition to the classroom [13]. The study found that students reported feeling significantly more attentive in class where active learning was used, and that there was no significant difference between satisfaction with student- and instructor-generated activities [13]. Cooperative learning has been used in a wide variety of subjects over many years, and some computer science educators have considered the use of specific roles for students to focus their attention on key concepts [4]. The authors of this particular study found that the benefits of cooperative learning outweighed any losses in learning due to reduced lecture time [4]. Peer-led team learning is a collaborative and active-learning technique that identifies certain students as peer leaders whose role it is to mentor and lead other students in creative problem solving [5]. In a qualitative study, the researchers showed marked and statistically significant improvements in student performance in a large-sized introductory computer science course [5]. Team-based learning has been used extensively in medicine since the 1980s, and a four-year study of team-based learning in a CS1 course found that it reduced attrition and improved student success [18]. Tools to promote active learning and collaboration in programming classes have also been developed. The Java Wiki Integrated Development Environment (JavaWIDE) is a particularly promising system that has been shown to promote active learning and collaboration in a heterogeneous set of teaching and learning environments [16].

Finally there is a study that considered the computer science classroom in general, and not just CS1 classes, that is of interest. Barker and Garvin-Doxas conducted an ethnographic research project in which they analyzed the learning environment in a selection of computer science classrooms at a large, research university in the United States [2]. Several of the classes were CS1, but the study included other computer science classes. They found that many of the approaches taken in the classrooms by instructors generated an impersonal environment that resulted in guarded behavior and the creation and maintenance of an informal hierarchy that enforced competitive rather than cooperative behavior [2]. They concluded with a case study demonstrating ways instructors can reduce the guarded and competitive behavior in students, including learning and using student names in order to decrease the distance between the instructor and students [2].

2.2 Commonalities

Although the pedagogical approaches surveyed in the previous section vary significantly in the details, there are some commonalities that exist with respect to student engagement:

1. Make the classroom dynamic: Pair programming, peer instruction, and the studio-based classroom all move the student from a passive observer into an active role. All of the pedagogical techniques discussed in this section are student-focused, which also contributes to a dynamic classroom for students.

2. Increase student-to-student interaction: Pair programming, peer instruction, peer-led team learning, and studio-based learning all require that students have significant interaction with each other, and this interaction is a significant contributor to the benefit of the approaches.

3. Reduce isolation: Students attribute some of the success of pair programming to their ability to rely on, and in some cases teach, other students. Techniques like pair programming, studio-based learning, collaborative learning, and peer instruction allow students to see that their classmates have many of the same issues that they do learning to program. These techniques also counteract the common misconception that computing is a field where most work occurs in isolation from other people.

4. Encourage a cooperative classroom environment: Many of the student-focused pedagogies include some form of cooperative learning, including pair programming, peer instruction, and studio-based learning. The ethnographic study of computer science classrooms suggests that this can be enhanced by specific instructor behaviors that reduce the distance between students and faculty, such as learning and using student names.

All of the techniques described in this section take a student-focused approach to teaching programming, and many of the techniques have been well-studied either in the context of computing or in related disciplines. However, short of adopting a
technique and designing a formal experiment, faculty who wish to use a novel pedagogical approach lack a way of evaluating their implementation. In the next section a behaviorist lens is applied to the commonalities found in the literature on CS1 pedagogy, demonstrating that student feedback can provide suggestions consistent with quantitative and qualitative studies on CS1 pedagogy.

3. LESSONS FROM STUDENTS
Student feedback can provide valuable information for CS1 faculty, both about the benefit of various pedagogical techniques as well as productive approaches for classroom management, contact with students, and ways to encourage student learning.

3.1 Background
The author has taught various introductory programming classes at the university level since 1993, but the information provided here is taken from courses taught in approximately the past five years. During that time the author worked as a professor at DePaul University. DePaul University is the largest Catholic institution of higher education in the United States, and all introductory programming courses are a part of the curriculum in the College of Computing and Digital Media (CDM). The student comments referenced here are taken primarily from CSC 241: Introduction to Computer Science I and CSC 242: Introduction to Computer Science II. The language taught in CSC 241/242 is Python. The population for CSC 241/242 consists only of undergraduates, and during the time in question, restricted to computer science majors or minors. All student comments are taken from private e-mails or from an anonymous survey conducted by a graduate researcher working with the instructor.

3.2 Student Comments
This section provides examples of CS1 student feedback and illustrates how that information can help instructors in evaluating their use of particular pedagogical approaches.

3.2.1 Dynamic Classroom
One of the most important ways that faculty can create a dynamic classroom is by encouraging student enthusiasm. Students who are excited about what they are learning will help to inspire other students to learn. Instructors who show students that they are excited about the material can make their students enthusiastic.

The following rave was sent by a student in response to some help provided on an assignment in CSC 242:

GUIs and I are in love again thanks to your help today!

Students who express love about the material are willing to work harder and create a positive atmosphere for their classmates. This particular student regularly makes comments in class about how interesting computer science is, which provides a positive impression of the course for his peers in the classroom.

Another important approach to cultivating a dynamic classroom is managing student frustration. Students taking their first programming class need to remain positive even while they are struggling to learn difficult concepts. The following quote illustrates an important attitude the instructor should take with respect to student frustration:

"Then I'll be able to explain my hatred for said problem before class, and we can attempt to get me to love it."

This student had indicated at the end of a class session that he hated one of the homework problems, a situation that needed to be remedied. By working with the student, an instructor can transform frustration into something more positive which will allow the student to remain engaged.

Another important consideration in CS1 courses is the interaction with non-majors. The focus of introductory computing courses is often on preparing majors for future classes, but non-majors can be an important audience. The following quote is from a student in CSC 241:

"The subject fascinates me and I would have loved to have majored in it, but my parents were very insistent on my going into the physical sciences."

Non-majors can provide a remarkable enthusiasm and different perspective on the material in CS1. Instructors need to be respectful of the contribution that non-majors can make in the classroom and encourage them to the same extent that they encourage students majoring in a computing field.

3.2.2 Student Interaction
Although it is shown in the literature on CS1 that student-to-student interaction has significant benefits for learning, it should be noted that students are not always pleased to be required to interact with other students. In the CSC 241 course a portion of class time is spent on a series of peer instruction activities. Students were surveyed about their reaction to the project in the Winter quarter 2012, and in response to the question “what did you like least about taking part in this project?” a student made the following comment:

"I didn’t like some of the questions, and I don’t like talking to people."

Although this feedback on the surface is negative, that is likely because it pushed the student beyond her comfort zone. Challenging students to move beyond computing stereotypes is not always comfortable for them, but it can be worth it. An instructor who is attuned to negative attitudes about novel pedagogical techniques can improve student impressions of the approaches, for example by mentioning the benefits of the activities.

3.2.3 Reduce Isolation
One of the most significant issues that students in a first programming course face is the adjustment to college. Provided with more independence than many of them have previously experienced, it is all too easy for them to develop poor study habits and a disregard for the importance of class attendance. Attendance during the freshman year, and particularly in classes for the major, is important for a student to avoid a sense of isolation. While taking attendance can be troublesome for various reasons, it may be sufficient to make students understand that you value their input, as illustrated by this comment from a freshman in her second quarter of Python who was going to miss a class:

"I know not everyone sends you e-mails about absences as this is college now, but it's important to me that you know I'm not ditching."

"I know not everyone sends you e-mails about absences as this is college now, but it's important to me that you know I'm not ditching."
This student was not always active in the classroom, but she clearly internalized the message that her attendance was expected. She also formed close bonds with several of her classmates, in no small part because of her regular attendance.

Student isolation can be exacerbated by a lack of connection with an instructor. The following note came from a former Python student:

> I was wondering if you knew Java and would you mind me asking questions about it because I have a really hard assignment and I’m a little lost.

While it is encouraging that the student felt a strong enough connection with a former instructor to ask for help on material beyond the class, it is discouraging that he did not feel comfortable talking with his Java instructor about the questions. Poor communication between the Java instructor and the student resulted in the instructor losing valuable opportunities to reinforce the information provided in the classroom.

3.2.4 Cooperative Environment

A large part of ensuring that there is a cooperative environment in the classroom is reducing the distance between faculty and students. Students who are comfortable asking a question, which is at its heart a display of ignorance, will be able to make more progress with the material. The following segment from an e-mail indicates several important things about this process:

> So I typed out this long letter with a question, only to figure out the answer myself while typing it. BUT I thought you might find it humorous or want to use it as an example or whatever, so I’m sending it to you anyways.

This was part of a lengthy exchange with the student about a homework assignment in which he consistently began to formulate a question and in the process of doing so discovered the answer to his own question. Students in the CS1 courses are encouraged to apply the “one-hour rule” in which they should ask a question after one hour of concentrated work without progress on an issue. They are also specifically told that sending a question and then retracting it if they discover the answer on their own is fine and often occurs. That the student did this and then wanted to share his thought process is particularly revealing because it indicates a feeling of connection and two-way dialog with the instructor.

3.2.5 Multiple Approaches

Often the approaches taken to improve the atmosphere and learning in the introductory computing classroom cannot be separated. Using students’ names is one approach to decreasing the distance between the students and instructor, but another is to allow the students to use the instructor’s first name. The following segment of an e-mail from a former Python student illustrates this:

> Ok, I'll try to call you Amber! I find it weird when [other student] reminiscences about "Amber's class." I have to think about what she is talking about. I think it seems strange to me just because she is so young. [...] I wouldn't call another professor by their first name, but I also don't think I've ever sent another professor more than 1 (maybe 2) emails.

As this quote illustrates, not all students are comfortable with using a faculty member’s first name, but doing so conveys equality. The student remarking on the relative ages of all people involved highlights this. A cooperative atmosphere extends past the boundaries of the classroom, as is illustrated by the student’s comment about a lack of contact with other faculty outside of class. And, finally, the student in question clearly continue to interact after the conclusion of the quarter, which demonstrates that the bonds they formed during the class remain.

4. CONCLUSION

The CS1 classroom is one of the most studied environments in computing education. A host of researchers have shown that innovative pedagogical techniques such as pair programming, peer instruction, and studio-based instruction have promise to improve student learning and engagement. Many of these approaches share common features, such as making the classroom dynamic, increasing student-to-student interaction, reducing isolation, and producing a cooperative learning atmosphere. But faculty who wish to adopt these approaches or otherwise improve their CS1 classroom have few formal approaches for professional development. Student evaluations can provide feedback, but it is typically not formative and comes after the fact. This article makes the case that faculty wishing to adopt new strategies in the CS1 classroom can use a behaviorist approach in their evaluation of their implementation. Student comments often echo the recommendations seen in the CS1 literature and can provide faculty with constant feedback throughout the quarter. The feedback is not always positive, since students exposed to new pedagogical approaches may be forced past their comfort zone, but it can be used to gauge in a less formal way whether the interventions the instructor is making are having the intended effect. Faculty who closely listen to their students may find that they are able to learn about teaching programming from the CS1 learners themselves.

5. ACKNOWLEDGMENTS

Thanks to Randy Connolly for an idea that eventually became this article and to André Berthiaume for his feedback on an earlier draft of this work.

6. REFERENCES


