The Use of Evidence in the Change Making Process of Computer Science Educators

Davide Fossati
Computer Science Department
Carnegie Mellon University
dfossati@cmu.edu

Mark Guzdial
College of Computing
Georgia Institute of Technology
guzdial@cc.gatech.edu

ABSTRACT

This paper explores the issue of what kind of evidence triggers changes in the teaching practice of Computer Science educators, and how educators evaluate the effectiveness of those changes. We interviewed 14 Computer Science instructors from three different institutions. Our study indicates that changes are mostly initiated from instructors' intuition, informal discussion with students, and anecdotal evidence.

Categories and Subject Descriptors

K.3.2 [**Computer and Information Science Education**]: Computer science education

General Terms

Human Factors

1. INTRODUCTION

This paper investigates the question of what kind of evidence triggers and informs changes in the everyday teaching practice of Computer Science educators. As educators, we want to be constantly improving our practice. Success depends on identifying the opportunities or need for improvement, implementing appropriate changes, and then (iteratively) evaluating whether the change met the need.

Our community is productive in providing tools for implementing change. A number of innovative approaches for teaching introductory Computer Science have been designed, such as Beyond LEGOs [3], Media Computation [14], and TeachScheme [9]. Many software systems to support new teaching approaches have been developed, such as Scratch [23] and Alice [6], as well as tools such as algorithm visualization systems [25] and intelligent tutoring systems for Computer Science topics [20, 12].

Researchers are studying the factors that influence adoption of these teaching innovations [21, 22, 16, 8]: What leads a teacher to choose one kind of implementation versus another? These studies explore both catalysts and barriers to change. For example, Ni [22] reports external pressure, limited time, poor background of students, and conflicting views on desired learning outcomes as barriers to change. She also highlights that perceived benefits for students, well-defined pedagogical recommendations, and successful first-hand experience with new approaches are catalysts to innovation adoption. Thus, we know a good bit about innovative approaches, and what leads to their adoption.

In this study, we would like to understand the factors that influence teachers' decisions at a more microscopic level. The teacher has to make decisions about where there is a need for change, and whether the change is effective at meeting the need. For example, why would a teacher change a specific example, a homework assignment, or the format of a group project? Once a change is made, how does a teacher decide whether the change addressed the concern that initiated the change? If those decisions are not made well (e.g., a change is made to something that wasn't really broken, or a change is actually ineffective when judged successful), we are not actually improving practice when we make change.

Specifically, we are interested in understanding the role of *evidence* in the decision making process of instructors. Researchers have spent some effort in understanding and affirming the important role of evidence in higher-level educational policy and practice [5]. The use of formal evidence is also an essential component of accreditation programs such as ABET [1]. Less attention has been devoted to the use of evidence in the the design decisions in classroom.

Our study aims to provide an initial understanding of this issue in the context of Computer Science education. We interviewed 14 Computer Science instructors from three higher education institutions in the United States, and we extracted the recurring themes in the interviewees' answers. These answers suggest the predominance of instructors' intuition, informal discussion with students, and anecdotal reports as the primary evidence used to inform practical decisions.

2. METHODOLOGY

We interviewed 14 Computer Science instructors, most of them teaching in large research institutions in the Midwest. The background and range of expertise of the interviewees spanned across the majority of sub-disciplines in Computer Science. Some of the instructors were full-time lecturers, whereas others were mostly dedicated to research. The range of classes taught by these instructors also ranged from introductory CS courses for undergraduate students to advanced elective classes taken mostly by graduate students. All the interviews were audio recorded.

We asked the instructors to recall episodes in which they implemented a change in their instructional practice. In particular, we asked for examples in which the change worked well, and cases in which the change did not work as expected. Then, we asked them to speculate on why those changes worked well, or did not work as expected. Then, we asked what triggered the change, and what kind of evidence was used to determine that a change was needed. A detailed script of the interview questions can be found at *http://www.fossati.us/papers/cschange-interview.pdf.* All the instructors felt comfortable answering these questions, and provided detailed and insightful answers.

The audio recordings of the interviews were fully transcribed by a professional transcription service. The transcripts were analyzed by a single annotator (one of the authors of this paper) who iterated through the answers extracting, summarizing, aggregating, and counting the recurring themes. These themes were not specified in advance, and emerged entirely from the data. Because the analysis was carried out by only one annotator, we cannot report any measure of inter-coder reliability.

3. RESULTS

3.1 Inventory of changes

Each instructor reported at least one example of successful change and one instance of failure. The episodes ranged from the simple change of a textbook to the introduction of more sophisticated pedagogical strategies, such as peer-supported interaction. They are summarized in Tables 1 and 2. We report some of these episodes, as voiced by the instructors, in the following subsections.

3.1.1 *Success stories*

One of the instructors implemented peer-sharing activities in class after being exposed to this methodology himself in a workshop. In this quote, we see both the instructor's decision to implement an innovation, and how the instructor judged the innovation to be successful.

> T03: The person at the seminar was talking about pedagogy, ways that we teach. Typically, in a class, you know, the lecturer might ask a question to get the audience to participate, but this person did something different. He called it "Peer-share." He said, "I want you to turn to the person next to you, and I want you to ask them this question." I don't even remember what the topic was. But the exercise of peer-share really struck me because all of us got to vocalize. Then after that, he had a few people talk, but everybody got to participate. So, that really struck me.
>
> So, then I started doing that in my classes, and that's been very effective. [...] I think it worked from a practical sense that people were able to participate, speak out loud. So, they were interacting with the material with a different mode of interaction, rather than listening passively. Socially, it also helps because often students in Introductory Programming class have questions, but they're afraid to ask because they might appear ignorant. So, by asking the person

Table 1: Success stories

Teacher	Description
T01	Increased the number of examples in class
T02	Introduced short quizzes in class
T03	Introduced peer-share interactions between students
T04	Had students work with a real database system
T05	Presented more actual code in class
T06	Changed paper presentations in classroom into critical analysis of the papers
T07	Introduced a new class in software engineering
T08	Asked students to write a short evaluation essay after every homework assignment
T09	Introduced the JFLAP animator of finite state automata in a theory course
T10	Introduced short bursts of collaborative work in class
T11	Kept a consistent picture making links between two courses in the same sequence
T12	Switched to programming languages with better tools
T13	Started doing in-class design exercises, using props like cards to get students out of the seats and talk to each other, thinking up design ideas.
T14	Gave lab exercises to practice the programming constructs just explained

Table 2: Failures

Teacher	Description
T01	Wanted students to give presentations on various topics
T02	Asked for a teaching evaluation in the middle of the semester
T03	Developed a complicated example that students found boring
T04	Had students write term papers
T05	Reduced the number of quizzes
T06	Moved from open projects to constrained projects
T07	Removed Smalltalk from an object-oriented programming class
T08	Introduced audio-recordings of the lectures
T09	Changed the textbook
T10	Had students give project presentations
T11	Used both Fortran and Matlab in the same course
T12	Used Java in a course where you needed a language that requires the programmer to take care of more details
T13	Tried stepping through code in a very detailed way using PowerPoint in class
T14	Tried some peer-programming by letting students work on projects together

next to them if that person also doesn't understand, then they feel confident that it's a good question to ask the entire class.

The following example is an alternative implementation on the same line of enabling peer interaction. In both of these cases, the teacher is identifying a need for increased collaboration as a desired change, implemented a methodology designed to enable collaboration, and then evaluated whether the methodology succeeded.

> T10: One of the changes that I implemented, and it really worked well, and then carried it through all my classes is little bursts, like five-minute collaborative work in class. So, I after I explain something, I typically pose a question which is just a tiny extension of the concept that was just explained or a variation. I let the students sort of turn to a neighbor, shake his or her hand, and do five minutes of brainstorming.
>
> [. . .] In general, my teaching style is very interactive. So, I ask questions and actually expect answers from a class, even if it's a 50-people class. But the problem with asking questions of 50 people is that only one or two ever sort of engage and give answers, and I want the whole class to be engaged. I want the whole class to actually think for a second about the concept that's just been explained, and internalize it. The only way I see of doing it is if they actually have to work with somebody else, and they have to either explain or ask questions of somebody else.

A different motivation from collaboration was a desire for greater authenticity. Generally, instructors believe that greater authenticity leads to greater motivation. Another instructor reflected on the need to provide students with hands-on experience with real systems.:

> T04: When I was first handed a database textbook and told to go teach it, we were using Date, which is a very well-written book, but we didn't have an actual system for people to work with. Once we got one, people were more interested, more motivated then trying to make things really work. That was effective. That's something that I guess happens in every Computer Science classroom. People really need practical examples to understand what it is they're doing effectively at a certain time.
>
> Then I started dividing the course up into groups and having groups of people construct actual databases for real applications. That also turned out to be an improvement. It turned out not to be sustainable as the class got to be 50 or 60 people. So, I had to give it up. But I kept those applications so that people could at least work with a variety of applications. [. . .]
>
> I think it worked well because the students got greater understanding from hands-on usage, but also because I think the groups worked well because the people involved had to explain things to each other and argue with each other, and consider alternative implementations.

Sometimes the motivations for change are external to the instructor. The following instructor successfully integrated students' feedback on previous homework assignments in his design of the homework assignments in the same course in the next iteration:

> T08: The one significant change I did after my first offering in class was that after every programming homework, I expect students to submit an essay, you know of ten lines, telling me the highs and lows of the homework. So, I make them reflect on the homework and give it a difficulty rating between zero and ten. Then what were the parts they found difficult? Were there any problems in the homework handout description, and all kinds of things. So, I made them sit and reflect on the homework after they have finished it for some time.
>
> Then I collect all the essays and I read them. Then if there was some problem with the homework, the way things are worded and so on, I made the changes to the homework so that next time around —I can't go back and change it for that class— but next time around I can at least make that experience better. So, over time, I see that there's a little bit better description of homeworks. I also know where the problem areas in homeworks are, and try to address that in my lecture.

3.1.2 Failures

Sometimes things don't work as expected. A common factor, like in the success stories, is the desire to generate increased motivation among the students. The following story shows how even a well-thought example in class might prove boring to the students:

> T03: I developed this long example that I thought was interesting. But as I presented it to this class, very smart kids in the top high school here, I realized that it was boring to them. I realized that they were just doing the exercise just to get the answer and they didn't connect it to things that were more meaningful. It was not good. [. . .]
>
> It didn't work because there were no follow on questions. It didn't make them excited. It didn't make them want to do it again. It didn't make them want to keep it, and do it at home with their friends or their brothers or sisters. I think good pedagogy, and good training, good examples lead you to more, rather than being closed-ended.

A teacher might decide that an idea is good, but the particular implementation is flawed. In this example, the teacher was also responding to student requests, but felt that the result was a failure due to the technical implementation of what seemed to be a good idea:

> T08: So, there was a time when I was trying to experiment with audio recordings for my lectures. Basically, you know, people wanted it. I, therefore, recorded my lectures and tried to put

Table 3: Reasons for success of instructional changes

Reason	Frequency
Motivation	8 (57%)
Practice, hands-on	3 (21%)
Environment and tools	3 (21%)

it online. But without the proper context, you know, where I was in the slides and so on, it just didn't work. [...]

It did not work because I was not screen-casting. I think what's important there is to do the screen-casting, you know, where you record both the screen and the voice.

Teaching is highly contextualized. The same idea may fail with a different course, set of students, or specific teacher. In this example, the teacher has decided that the idea of "peer-programming" (or pair-programming [19]), which has been highly successful in the literature, failed in hes course because of the context of her students:

T14: I did at one point try to do a little bit of peer-programming. This was at my other job. So, I'd let people work on projects together. What wound up happening is one person would do the project, and the other person wouldn't be involved at all. So, I don't think that worked particularly well in that situation.

I'm doing the same thing in my current job though, and I think it works better just because the traditional students in there are around and they're able to work together, where I had adults that just weren't able to do that.

[...] That's why I think it didn't work is that they just weren't able to spend time working together.

3.2 Reasons for success and failure

In the previous section, we already saw some examples of how the instructors explained their success and failure stories. Overall, a fairly structured pattern emerged from the teachers' reflections on the causes of success or failure of their interventions (Tables 3 and 4). The responses were not mutually exclusive, so the percentages reported in the tables do not necessarily sum up to 100%.

The majority of instructors credited the success of the change to motivational factors such as improved student participation and interaction; positive support for creativity and confidence; variety; and more appeal to the students. Another success factor mentioned by some of the instructors is the increased engagement of students in hands-on activities, such as programming assignments or projects dealing with interesting real-world problems. Other teachers mentioned environmental factors such as better programming environments, useful tools, and effective visualizations, as reasons for success.

Blame was attributed to the background of students; motivational issues; organization or presentation of the information; and technical or logistical issues.

Table 4: Reasons for failure of instructional changes

Reason	Frequency
Background of students	4 (29%)
Motivation, appeal to students	3 (21%)
Presentation, organization of the material	3 (21%)
Technical, logistical issues	2 (14%)

Table 5: Evidence that triggered the change

Response	Frequency
Informal discussions with students; anecdotal reports; reaction of students	9 (64%)
Reflections of the instructor; feeling, perception, intuition	9 (64%)
Bad student performance; test results	4 (29%)
Perceived a need for constant change	3 (21%)
Teaching evaluation from students	2 (14%)
Opinion of the department, other teachers	2 (14%)
Research results, literature	2 (14%)

3.3 Triggers of change

When asked about what triggered the change and what kind of evidence was used to determine that a change was needed, the instructors elaborated on several reasons, summarized in Table 5. The responses were not mutually exclusive, so the percentages do not sum up to 100%. Overall„ the answers revealed that some kind of informal evidence was used, such as informal discussion with some students — maybe just the top students, as one instructor explicitly commented, implying a particular focus or value from the instructor:

T12: Oftentimes, there's a set of the top students in there. You know, you build up a rapport and you start talking to them about different stuff. In this, they give you kind of a feedback. They'll say, "Well, you know, I just didn't understand why we covered this. It seemed like a waste of time." It's like, okay.

Some instructors verbalized the conflict between a perceived —or requested— need for formal evidence, and the real driving factors:

T13: Well, I would love to say that I was carefully monitoring the students the whole time, and once a certain percentage became bored, I switched over. But really it was just the dictates of the curriculum. I mean, I could tell, you know, that the lectures weren't the best thing to use.

T12: But it almost always seems that you feel the need is there. Then it's almost with the ABET stuff that you almost have to go about and, "Okay. We feel the need. We want to make this change. Oh, look! Here's a statistic that we can use to prove this." Okay. We're just pulling that statistic in just to, "Okay. Good. We can check off the ABET box and said we did it in there."

Sometimes it was a combination of anecdotal evidence, teaching evaluations, and general observation of student performance, with informal evidence being the main reason:

> T05: It's really anecdotal mostly, I guess. But student performance seemed to degrade a little bit, a few comments from evaluations that indicated that there was a feeling that the structure wasn't as strong. Of course, these weren't students that had taken it in the other form, but this is my inference. But it's a little anecdotal.

Some instructors felt that there is an explicit need for constant change in teaching Computer Science:

> T07: Well, in Computer Science you always think about change. Things change all the time. So, you cannot teach the same course, except for certain materials.

Sometimes external forces, such as pressure from the Department, played a role into the instructors' decisions:

> T11: That one was based upon my opinion. Part of it was from the Department saying, "Hey, we want more MATLAB," and part of it was I thought my experience was that these other disciplines, the syntax of any one language is not the problem.

Only a few instructors mentioned test scores and student performance as their primary source of evidence. But even when evidence was cited, it was not often used to target a particular need to change:

> T14: My first exam. They did not do very well. So, that's self-explanatory.

4. DISCUSSION

Education is a designed activity, and can be studied like any other design activity [24]. When they make decisions, teachers are engaging in a design process, specifically, an instructional design process [13]. Specifically, some researchers have talked about creating a science of studying educational design decisions [7]. To interpret and understand the data presented in this paper, we propose to think about an ideal decision-making design process of instructors as composed of three parts:

1. Making a determination that a change is needed.
2. Either finding existing solutions or creating new interventions to address the desired change.
3. Evaluating the effectiveness of the solution and deciding whether to retain it or not.

To date, mostly the second step of this process benefits from abundant research in Computer Science Education, as it is shown by the number of good papers appearing in professional meetings such as the ACM SIGCSE. More and more resources are becoming available to Computer Science educators, such as in the Computing and Information Technology Interactive Digital Educational Library (CITIDEL) [15] and the Ensemble project [4].

However, we argue that taking into account the first and the third step of the design pipeline is essential if we want to ensure that teaching innovations are adopted by practitioners in a sustainable way. As in any serious science and engineering design task, we don't just want good components: we also want good decisions grounded in evidence, and good evaluation and quality control. Without a careful determination of needs and requirements (step 1), we may end up "fixing the wrong thing," implementing changes that are not really needed, and overlooking areas that would need real improvement. Without a careful quality control (step 3), we may attribute the outcome of an intervention to the wrong causes, or fail to recognize the influence of the implementation context to the specific interventions.

The *Disciplinary Commons* [26, 11] is an approach that addresses the first step of the pipeline. In this model, the practice of Computer Science teachers is documented and shared among peers, making this information public and available for future use. By reflecting on their practice, and comparing their practice to others, the teachers engage in a process of review that can highlight areas in need of change in their practice. A Disciplinary Commons approach can be effective, but is expensive (in terms of teacher time). Empirical measures may be more useful for the individual teacher working on his or her own.

For the third step of the design pipeline, we note that Multi-Institutional, Multi-National (MIMN) studies [18, 17, 10] offer a way of comparing results across institutions, and thus, creating benchmarks that inform our decisions about effective practice. If students do better in my class than on problems from these studies, I have effectively compared them to all those institutions. The MIMN studies themselves are teachers working collaboratively to pool data to facilitate analysis and provide greater insight. It is still not common practice, however, for regular instructors to "scale down" this process to their own classrooms. Further, the scale of MIMN studies prevents teachers from using them for informing microscopic decisions. How do I decide that *that* particular homework assignment worked, or didn't?

Within the limitations of size and scope of this study, our results suggest that Computer Science instructors rely mostly on intuition and anecdotal evidence to make decisions about changes in their daily teaching practice. We are not arguing that the teachers' decisions are bad ones. In fact, our interviews show several examples of (what seemed to us to be) good decisions. All we know is that instructors used little empirical data for deciding to make a change, and for deciding whether a change was successful or a failure.

Why aren't instructors using more empirical evidence? We can formulate two alternative hypotheses. On the one hand, instructors may not want to use this kind of evidence. Basing day-to-day decisions on empirical evidence might take too much time to be practical. At this level, some decisions should be made rather quickly, and the overhead required by a formal data analysis might paralyze the decision-making process. Additionally, some forms of data analysis would require methods and skills outside the expertise and interests of most instructors.

On the other hand, data may simply not be available, or it may be difficult to access. Available data may have been collected for different purposes, and it might not contain the exact information needed to make meaningful decisions. For example, exams and homework assignments are usually graded in a way that facilitates summative evaluation of student performance, and just the bare grades are usually not detailed enough to inform specific teaching interventions.

A promising way to alleviate this problem is the development of technology that can automate the analysis of educational data collected during regular classroom practice, and support the decision-making process of instructors by extracting actionable information from such data. Researchers in the emerging field of Educational Data Mining [2] are actively working on such techniques and applications.

5. ACKNOWLEDGMENTS

This work is supported by the Computing Research Association (Computing Innovation Fellowship CIF–186 under NSF Grant 0937060), and the National Science Foundation (Broadening Participation in Computing Award 0940394).

6. REFERENCES

[1] ABET: Leadership and quality assurance in applied science, computing, engineering, and technology education. http://www.abet.org/.

[2] R. Baker and K. Yacef. The state of educational data mining in 2009: A review and future visions. *Journal of Educational Data Mining*, 1(1):3–17, October 2009.

[3] D. Blank, L. Meeden, and D. Kumar. Python robotics: an environment for exploring robotics beyond legos. In *SIGCSE '03: Proceedings of the 34th SIGCSE technical symposium on Computer science education*, pages 317–321, New York, NY, USA, 2003. ACM.

[4] L. Cassel, H. Gregory, and B. Nadella. Ensemble: Enriching communities and collections to support education in computing. In *ITiCSE 2009, 14th Annual Conference on Innovation and Technology in Computer Science Education*, page 355, Paris, 2009.

[5] A. Cooper, B. Levin, and C. Campbell. The growing (but still limited) importance of evidence in education policy and practice. *Journal of Educational Change*, 10(2–3):159–171, May 2009.

[6] W. P. Dann, S. Cooper, and R. Pausch. *Learning to Program with Alice*. Prentice Hall, 2005.

[7] A. Disessa, M. Gardner, J. G. Greeno, F. Reif, A. H. Schoenfeld, and E. Stage. *Toward a Scientific Practice of Science Education*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1990.

[8] R. Donnelly. Exploring lecturers' self-perception of change in teaching practice. *Teaching in Higher Education*, 11(2):203–217, 2006.

[9] M. Felleisen, R. B. Findler, M. Flatt, and S. Krishnamurthi. *How to Design Programs: An Introduction to Computing and Programming*. The MIT Press, 2001.

[10] S. Fincher, R. Lister, T. Clear, A. Robins, J. Tenenberg, and M. Petre. Multi-institutional, multi-national studies in CSEd research: some design considerations and trade-offs. In *ICER 2005, First International Workshop on Computing Education Research*, pages 111–121, Seattle, WA, 2005.

[11] S. Fincher and J. Tenenberg. Warren's question. In *ICER 2007, Third International Computing Education Research Workshop*, pages 51–60, September 2007.

[12] D. Fossati, B. Di Eugenio, C. Brown, S. Ohlsson, D. Cosejo, and L. Chen. Supporting computer science curriculum: Exploring and learning linked lists with iList. *IEEE Transactions on Learning Technologies, Special Issue on Real-World Applications of Intelligent Tutoring Systems*, 2(2):107–120, May 2009.

[13] R. M. Gagné, L. J. Briggs, W. W. Wagner, K. Golas, and J. M. Keller. *Principles of Instructional Design*. Wadsworth, 5th edition, 2004.

[14] M. Guzdial and B. Ericson. *Introduction to Computing and Programming with Java: A Multimedia Approach*. Prentice Hall, 2006.

[15] D. Knox. Citidel: Making resources available. In *ITiCSE 2002, 7th Annual Conference on Innovation and Technology in Computer Science Education*, page 225, Aarhus, Denmark, 2002.

[16] R. B.-B. Levy and M. Ben-Ari. We work so hard and they don't use it: Acceptance of software tools by teachers. In *ACM SIGCSE Bulletin*, volume 39, pages 246–250, 2007.

[17] R. Lister, E. S. Adams, S. Fitzgerald, W. Fone, J. Hamer, M. Lindholm, R. McCartney, J. E. Moström, K. Sanders, O. Seppälä, B. Simon, and L. Thomas. A multi-national study of reading and tracing skills in novice programmers. In *ITiCSE-WGR '04: Working group reports from ITiCSE on Innovation and technology in computer science education*, pages 119–150, 2004.

[18] M. McCracken, V. Almstrum, D. Diaz, M. Guzdial, D. Hagan, Y. B.-D. Kolikant, C. Laxer, L. Thomas, I. Utting, and T. Wilusz. A multi-national, multi-institutional study of assessment of programming skills of first-year CS students. *SIGCSE Bulletin*, 33(4):125–180, 2001.

[19] C. McDowell, L. Werner, H. E. Bullock, and J. Fernald. Pair programming improves student retention, confidence, and program quality. *Communications of the ACM*, 49(8):90–95, 2006.

[20] A. Mitrović, P. Suraweera, B. Martin, and A. Weerasinghe. DB-suite: Experiences with three intelligent, web-based database tutors. *Journal of Interactive Learning Research*, 15(4):409–432, 2004.

[21] L. Ni. What makes CS teachers change? factors influencing CS teachers' adoption of curriculum innovations. In *SIGCSE 09, The 40th ACM technical symposium on Computer science education*, pages 544–548, Chattanooga, TN, 2009.

[22] L. Ni, T. McKlin, and M. Guzdial. How do computing faculty adopt curriculum innovations? the story from instructors. In *SIGCSE 10, the 41st ACM technical symposium on Computer science education*, pages 544–548, Milwaukee, WI, 2010.

[23] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, and Y. Kafai. Scratch: Programming for all. *Communications of the ACM*, 52(11):60–67, 2009.

[24] H. A. Simon. *Sciences of the Artificial*. MIT Press, Cambridge, MA, 1996.

[25] J. T. Stasko and C. D. Hundhausen. Algorithm visualization. In S. Fincher and M. Petre, editors, *Computer Science Education Research*, chapter 6, pages 199–228. Routledge, London, 2004.

[26] J. Tenenberg and S. Fincher. Opening the door of the computer science classroom: The disciplinary commons. In *SIGCSE 2007, 38th SIGCSE technical symposium on Computer Science Education*, pages 514–518, Covington, KY, 2007.