Experience Report: A Multi-classroom Report on the Value of Peer Instruction

Leo Porter, Cynthia Bailey Lee, Beth Simon
Computer Science and Engr. Dept.
University of California, San Diego
La Jolla, CA USA
+1 858 534 5419
{leporter,clbailey,esimon}@ucsd.edu

Quintin Cutts
Sch. of Computing Science
University of Glasgow
Glasgow, Scotland
+44 141 330 5619
quintin@dcs.gla.ac.uk

Daniel Zingaro
Dept. of Computer Science
University of Toronto
Toronto, ON, Canada
daniel.zingaro@utoronto.ca

ABSTRACT

Peer Instruction (PI) has a significant following in physics, biology, and chemistry education. Although many CS educators are aware of PI as a pedagogy, the adoption rate in CS is low. This paper reports on four instructors with varying motivations and course contexts and the value they found in adopting PI. Although there are many documented benefits of PI for students (e.g. increased learning), here we describe the experience of the instructor by looking in detail at one particular question they posed in class. Through discussion of the instructors' experiences in their classrooms, we support educators in consideration of whether they would like to have similar classroom experiences. Our primary findings show instructors appreciate that PI assists students in addressing course concepts at a deep level, assists instructors in dynamically adapting their class to address student misunderstandings and, overall, that PI encourages students to be engaged in conversations which help build technical communication skills. We propose that using PI to engage students in these activities can effectively support training in analysis and teamwork skills.

Categories and Subject Descriptors

K.3.2 [**Computer Science Education**].

General Terms

Algorithms, Human Factors

Keywords

CS1, peer instruction, clickers, PRS, classroom response, active learning.

1. INTRODUCTION

Peer Instruction (PI) is a pedagogy that involves a conceptual shift away from a classroom dominated by one-way transmission of knowledge to a more collaborative, active learning environment [4]. For the past twenty years, PI has contributed to improvements in conceptual understanding among physics students [4], and is now being adopted by educators of other sciences [2,7]. Yet, adoption of PI in CS remains slow, with few reporting on its use in the last few years [5,9,10,13].

Traditionally, experience reports describe an instructor's experience with one course or tool—reporting their adoption experience. Here, rather than reporting on one course, we report the experience of adopting one *pedagogy* – across multiple instructors, institutions, and computing courses. Four instructors who adopted PI for varying reasons and in varying courses[1] each describe one question which was particularly compelling in their situation. From these experiences, we offer computing educators in many courses the opportunity to consider whether these reflect experiences they would like to have in their classrooms.

We do not report on the impact of PI on student learning here. Using a standardized concept inventory (CI) for comparison, PI has been shown to increase learning approximately twofold in large studies of physics courses [4,6]. It is difficult to make such claims in CS, where we have no accepted CIs available through which to measure learning gains. Instead, both the extensive documentation of PI's effect on learning in other disciplines and the clear basis of the pedagogy in constructivist learning models argue that it would be effective in computing.

Additionally, we do not fully document the process of developing and deploying a course using the PI pedagogy. There are numerous resources available that describe this process, for example, the video series and interdisciplinary handbook [12] and tactics for generating effective MCQs [1]. Closer to home, an experience report on adopting and adapting PI in a CS1 course can be found in [10]. We will provide discussion across the four experiences and describe how we believe PI naturally supports development of effective analysis, teamwork, and communication skills.

2. BACKGROUND & RELATED WORK

PI is a classroom pedagogy that makes use of clickers to enable all students in a class to respond or vote on multiple choice questions (MCQs) posed by the lecturer. A PI classroom session involves several iterations of a well-defined vote/discuss/re-vote procedure. Each PI iteration begins with an MCQ designed both to focus attention on and raise awareness of a key course concept. After individually thinking about and voting on the correct answer (the individual vote), students discuss the question in small groups, reach a consensus, and vote again on the same question (the group vote). The instructor can then make use of the results of both votes to lead a class-wide discussion to help review persistent misunderstandings. In the small groups, students are encouraged to discuss each answer option, verbalizing why it is correct or incorrect [10]. The discussion typically results in more students giving the correct answer, and there is evidence that this

[1] CS0, CS1, theory of computation, and computer architecture

improvement reflects gains in conceptual understanding rather than peer influence [11]. This process is often accompanied by quizzes of prior reading and mini-lectures during class. The reading quizzes, completed before class, prepare students to engage with in-class MCQs. The mini-lectures are given before particularly challenging PI questions to further prepare students.

The seminal work on PI reviewed learning gains achieved using PI in introductory physics over the span of 10 years [4]. Increases in conceptual understanding, measured using standardized tests applied before and after the courses, showed approximately a two-fold improvement over traditional methods.

In other disciplines, instructors have provided personal accounts of how the use of PI and clickers have changed their teaching [2,7]. In computing, [5,9,10,13] have discussed the process of integrating PI in their classrooms; [3] reports on brief uses of PI in an otherwise traditional course. Of these studies, [10] is the only one to include instructor reflections; for example, the instructor found that generating distracters for MCQs helped her make student misconceptions more explicit.

3. Theory of Computation

We next present and discuss four PI questions, one per section. The questions in this section and Section 4 demonstrate that students collectively learned via effective discussion. Sections 5 and 6 contain questions where student discussion helped the instructor identify problems and dynamically adjust course content to address misunderstandings.

Our first example comes from a Theory of Computation (Automata) course conducted in the summer of 2010 at a large R1 institution in the US. The course is limited to Computer Science majors, and is required for the major. The course seeks to develop an understanding of the mechanics of automata types (Discrete Finite Automata, Pushdown Automata, Turing Machines) and the properties of related language classes (Regular Languages, Context-Free Languages), as well as refine proof-writing skills.

Lectures were conducted almost entirely in PI format. A primary goal of using PI for this course was to give students as much practice as possible in the clear, concise, and convincing communication of theoretical concepts. Such communication is the essence of proof-writing, but is a fundamental challenge for many students. Proof-writing is not a new activity for students, but application of an understanding of what constitutes a proof is typically inconsistent. For example, even in this upper-division course, it was not uncommon for the instructor to observe students attempting to prove a universal statement by analyzing one specific example (a classically fallacious approach). Students can overcome these challenges with extensive practice and feedback. However, proofs are highly labor-intensive to read and assess, making it logistically difficult for course staff to provide adequate formative feedback to students.

PI provides an early, interactive environment for practicing communication of theoretical content, and, in some cases, even composing complete arguments. Compared to a traditional written assignment, classroom discussion is actually a more natural context for the act of proving. Proving is supposed to be convincing a peer, not merely producing a "correct" artifact in a vacuum or for a professor who one knows is already trivially convinced.

Many of the MCQs used in the course focused on automata mechanics and design. With these, PI develops essential skills in

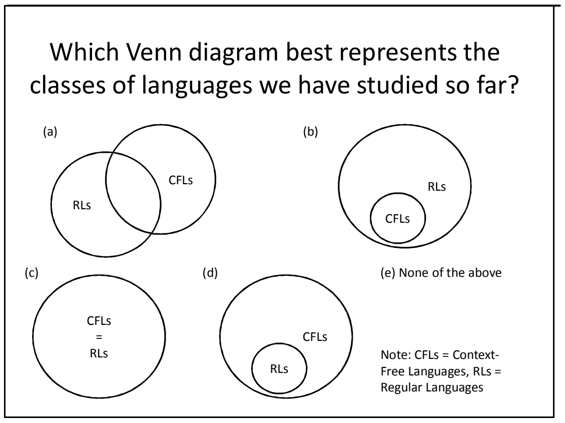

Which Venn diagram best represents the classes of languages we have studied so far?

Figure 1: Language Classes MCQ

naming theoretical objects precisely, and analyzing their properties in words. These will be the building blocks for precise proof-writing. Other slides directly ask students to complete a proof, for example by selecting the correct missing sentence from a partially-completed proof. The MCQ presented here is one of a few from the course that does not directly ask students to construct a proof, yet induces spontaneous proof arguments during discussion.

3.1 Question: Language Classes

The question in Figure 1 is presented following PI slides introducing Pushdown Automata (PDA), which are associated with the class of Context-Free Languages (CFLs). Thus students are newly familiar with the mechanics of Pushdown Automata, but this is the first question on the properties of their class of languages, the CFLs. Students have previously studied the less-powerful Nondeterministic Finite Automata (NFA), and associated class of Regular Languages (RLs). The correct answer to this question is (d): the class of Regular Languages is a strict subset of the class of Context-Free Languages.

3.2 Results

The voting for this question was 26% (a), 67% (d), and 3% each of (b), (c), and (e). It would appear that most students began with the notion that some non-Regular languages are Context-Free, probably from experience with analyzing Pushdown Automata for languages that are clearly not Regular in the preceding slides. This understanding narrows the compatible choices to (a) and (d). So the key was to move students from (a) to (d).

Moving students from (a) to (d) involves proving that there is no RL that isn't also a CFL, in other words, that the class of RLs is a subset of the class of CFLs. To do this, we observe that, given an NFA, one can construct a PDA for the same language---simply copy the NFA's structure and ignore the additional stack feature of PDAs. Just as the instructor had hoped, she observed some students spontaneously invoking this argument in their group discussions.

In the whole-class discussion that followed, a few less rigorous explanations were offered along with the one above. It served as a good case study for how to identify arguments that are unambiguously convincing to people who are uncertain of the outcome, and motivated a review of a standard "toolkit" of approaches to proving different Venn diagram configurations. Peer Instruction was successful in inducing the exact kinds of natural context argumentation experiences and subsequent analysis that the instructor had desired for this course.

4. Computer Architecture

The following example appeared in an Introduction to Computer Architecture class in the summer of 2010 taught at a large R1 institution in the US. The class was taught using PI, clickers, and online reading quizzes before each class.

Computer Architecture poses particular challenges for the PI instructor. Similar to physics, many of the questions that appear in the text can be solved by applying a formula or basic algorithm to obtain the correct result. Examples of such problems include standard plug-and-chug problems using the performance equation, detailed decoding/encoding of MIPS assembly instructions, etc. However, these problems often fail to address core class concepts. For example, although encoding/decoding of MIPS instructions helps to clarify the role of an assembler, such questions may not drive at core concepts in Instruction Set Architecture design.

Although many of the PI questions from the class are interesting and many offer larger gains in student performance between individual and group votes, this particular question was chosen because of the discussion it fostered on core class concepts.

4.1 Question: Pipeline Design

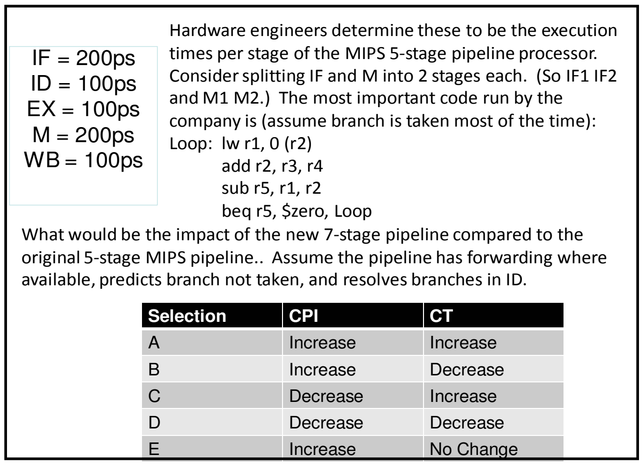
Figure 2 is a question on pipeline design. When teaching pipelining in an undergraduate course, students often become so mired in the details of hazard detection logic, data forwarding, branch prediction logic, etc. that they lose sight of the big picture. This question was asked at the end of the section on pipelining to refocus students on the big picture.

In this question, splitting instruction fetch (IF) and memory (M) into two stages each will increase the impact of hazards on cycles-per-instruction (CPI). The students had worked with examples with a similar 7-stage pipeline so they had enough background that they could be capable of recognizing that longer pipelines tend to increase the impact of hazards on CPI. Similarly, they had worked examples in multi-cycle processors on the impact of properly balancing the amount of work done per cycle. In this problem, splitting IF and M into two stages better balances the time spent per stage of the pipeline and allows us to reduce cycle time (CT) by a factor of two. The correct answer for this question is (b).

4.2 Results

The initial voting for this question was 5% (a), 61% (b), 20% (c), and 7% each of (d) and (e). After the group vote, a larger number of students answered correctly (61% to 71%) but there were still 29% confused about the result. Of those confused about the correct result, 5% answered (c) and 12% answered each of (d) and (e). The complete opposite of the correct answer (increase in CT but decrease in CPI) was response (c). Considerably fewer students responded (c) after the group discussion. Both (d) and (e) were half correct in that (d) had the correct response for CT but the wrong response for CPI and (e) had the correct response for CPI but not CT. Thus, the improvement in their group response was not just in more students answering entirely correctly, but more students answering partially correctly.

Following the group vote, the instructor led a class-wide discussion addressing the correct response for CPI and CT indiv-



IF = 200ps
ID = 100ps
EX = 100ps
M = 200ps
WB = 100ps

Hardware engineers determine these to be the execution times per stage of the MIPS 5-stage pipeline processor. Consider splitting IF and M into 2 stages each. (So IF1 IF2 and M1 M2.) The most important code run by the company is (assume branch is taken most of the time):

```
Loop:  lw r1, 0 (r2)
       add r2, r3, r4
       sub r5, r1, r2
       beq r5, $zero, Loop
```

What would be the impact of the new 7-stage pipeline compared to the original 5-stage MIPS pipeline.. Assume the pipeline has forwarding where available, predicts branch not taken, and resolves branches in ID.

Selection	CPI	CT
A	Increase	Increase
B	Increase	Decrease
C	Decrease	Increase
D	Decrease	Decrease
E	Increase	No Change

Figure 2: Computer architecture question regarding the impact of longer (deeper) pipelines on CPI and CT.

idually. Finally, after discussing the correct response, the instructor used this example to segue into why deeper pipelines can be attractive and a longer explanation of tradeoffs involved in deciding on pipeline depth (namely its impact on CPI and CT).

What the instructor was most impressed about was that, led by a need to answer this question, students discussed high-level course content in a lively and intelligent manner. Having worked with students from prior classes for a number of years, the instructor had previously complained that students *rarely* have solid discussions on core course content. In prior years, students often focused on the aforementioned details of pipelining while missing the big picture (as demonstrated in exams and in conversations with the instructor). The few occasions where students were concerned about these tradeoffs seemed to only occur a few days (or even hours) before major exams. However, because of PI and the design of this question, students spent at least a significant period of class time engaging core concepts (in detail, with lively debate). Also, because this discussion occurred in class, the instructor was available to assist with their learning by guiding the resulting class-wide discussion. What had previously been an issue for “cram” sessions became an integral and vibrant part of the class experience.

5. CS0

This example comes from a large (~570 student) CS0 course, which was one pilot course for the AP CS Principles program in Fall 2010 at a large R1 institution in the US. The course is a required general education elective for some undergraduates, and seeks to develop computational thinking skills among students who may never take another computing course. The course focuses on developing technical analysis and communication skills through instruction in Alice and Excel. PI was a core feature of the instructional design of the course. We chose to use PI because it allowed a significant amount of students’ time engaging in the course to be devoted to analysis and communication practice. Although we were “teaching” students to program in Alice, the real goal of the course was to affect students’ confidence in using computers, positively impact their views of technology as a value in society, and develop their skills in “dealing” with computers and computation through analysis, communication, and organization skills. As such, most of the MCQs asked students to do one of the following:

- select a line of code to complete a given program to make it do a described task
- select a correct English description of what a code does
- identify the appropriate rationale for why a code works the way it does

5.1 Question: Dynamic Execution

Figure 4 shows an example of a "rationale" question asked in class. The question is referring to a line of code that students had just practiced evaluating (for specific values which resulted in 0.35) in the previous question. Up to this point, students had no specific experience in evaluating numerical expressions. Having just performed an evaluation given a specific set of objects, the concern that students do not recognize that the expression can evaluate to different values during different executions is raised. The core issue is to illuminate the key feature of the static code - dynamic execution model of programming.

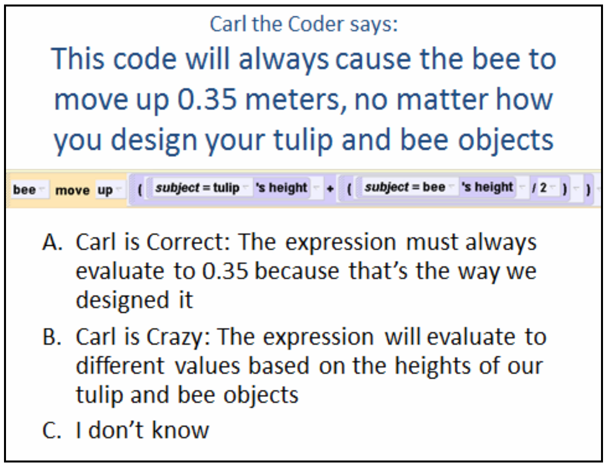
5.2 Results

When students considered this question individually, 89% answered correctly. After group discussion this increased to 98%. However, with this style of question, student correctness in voting is only the first step in valuable learning from the problem. The student-provided explanation during the class-wide discussion was "we didn't specify how tall the tulip and the bee are, and, yeah, it can change." This not-quite-satisfactory explanation both illuminated for the instructor students' interpretations of how dynamic execution happened and provided the opportunity for her to model her own way of thinking about the various ways in which the code could be called. This then led naturally into a demo of a situation where three tulips of different heights were used. We called the code three different times and it produced different values each time. The instructor was able to help students clarify and hopefully understand a more expert explanation – it is not that we do not specify a height, it is that the height can vary during various dynamic executions of the statement.

Again, the real value of this question cannot be summed up in the student voting record alone. The primary goal of the question was to engage students in discussions in their own words that can help them identify the static/dynamic execution concept. The issue is very abstract (yet core) and even though a demo where the steps through instructions are highlighted may help, here we use PI to help prepare students to learn a concept by having them engage in it first. The value of this approach in PI has been documented both in giving students appropriate time to consider an issue themselves and "try out" their understanding in discussions and in providing an impetus for students to want to focus and understand the following explanation and demonstration.

6. CS1

This section's example comes from a small (40 students), first-year, remedial CS1 course for engineering students at a large Canadian research university. All students in the course had been unsuccessful in taking the course the semester prior, and the course is required in order to continue in their program. PI was faithfully used: each class was structured around three or four MCQs, with reading quizzes graded for completion prior to each class. The course used C to teach standard introductory CS topics.



Carl the Coder says:

This code will always cause the bee to move up 0.35 meters, no matter how you design your tulip and bee objects

```
bee move up ( subject = tulip 's height + ( subject = bee 's height / 2 ) )
```

A. Carl is Correct: The expression must always evaluate to 0.35 because that's the way we designed it

B. Carl is Crazy: The expression will evaluate to different values based on the heights of our tulip and bee objects

C. I don't know

Figure 4: Question on dynamic execution in CS0.

Consider the following code.

```
void recur (int i) {
  if (i == 0) {
    printf ("%d ", i);
    return;
  }
  for (int j = 0; j < 2; j++)
    recur (i - 1);
}
```

What is printed by the call recur (1)?

A. 0
B. 0 0
C. 0 0 0 ... infinitely
D. 0 1
E. None of the above

Figure 5: Recursion question in a CS1 class.

In CS1, it is tempting to ask "factual recall" MCQs ("what is the C construct for creating a loop?"), as these are the concepts introduced in the course. Yet such questions can hardly be expected to generate animated discussion or student anticipation. Instead, many of our questions applied core constructs to determine whether students really understand how they work in practice.

6.1 Question: Recursion

We used the example here (Figure 5) in the third lecture on recursion. The first recursion lecture introduced the terminology and concepts of recursive flow, and the second provided several typical examples of recursion (reversing a number, finding the length of a string, etc.). We intended the example given in Figure 5 to be a quick warm-up that would lead directly into a discussion of recursive backtracking algorithms. As in any good PI question, the distracters target common misconceptions: answer (a) targets a looping mental model of recursion [8], (c) seeks a misunderstanding of the base case, etc. However, nothing in this question was meant to be tricky, or even particularly challenging.

6.2 Results

The supposed "warm-up" question showed that students were not yet ready to proceed further. The individual vote yielded a correctness of 50%; the group vote, hampered by this paucity of student understanding, yielded a correctness of 48%. Rather than proceed with recursive backtracking, undoubtedly further distancing at least half of the students, the instructor decided to spend the rest of lecture on questions similar to that given in Figure 5. It took two further MCQs, with careful discussion and tracing, until the instructor felt that students understood the concept well enough (69% correctness on a group vote) to proceed with backtracking in the next class. Without the use of this MCQ, the instructor would have proceeded to backtracking too early, being rather out of touch with students' level of understanding. PI often gets lauded for its ability to engage students in discussion and improve their conceptual understanding. We should also keep in mind its ability to help create flexible, demand-driven class meetings, where the demands come directly from student responses.

7. DISCUSSION

The examples discussed here show instructors finding value in PI in the ways that it:

- Enables instructors to dynamically adapt class to address student misunderstandings
- Engages students in exploration and analysis of deep course concepts
- Explores arguments through team discussions to build effective, appropriate communication skills

Through discussions among the authors, we identified experiences not reported in other disciplines. In many PI studies, deep understanding is the primary goal with student discussion a means to that goal. In computing, we believe PI both improves student understanding *and* fosters teamwork and communication skills. In our instructors' experience, the communication skills fostered through PI moved to the foreground.

Clearly, one of the benefits our instructors value is the ***ability of PI to support students in explicit training of analysis and argumentation skills***. The focus of much of computing education is on producing: write a program, build a model, or develop a proof. One of the benefits of teaching computing is that we often rely on students' experiences with "programming projects" to foster deep understanding of concepts (hence OS simulators, processor design projects, etc.). However, common practice rewards the completion of (working) projects and products. It rarely evaluates the process and even more rarely assesses the analysis or understanding underlying those works. PI seems a natural model for embedding such desirable learning goals into existing coursework. While the classroom experience is a dramatic change from standard lecture formats, it is easy to implement in large scale, standard-seating arrangements, and does not require dramatic change to the course content.

Helping computing students develop teamwork skills is critical. ***PI dramatically increases the emphasis on teamwork and communication skills.*** Although different courses and instructors will vary, students in PI classrooms anecdotally spend 30-50% of lecture periods in small-group discussion, with additional time spent listening to the class-wide analysis and discussion. While we know of no examples where students doing PI are directly evaluated on their discussions in class, it is not challenging to imagine various ways it could be done.

Both of these findings lead to the single most important and simplest advice we have for instructors developing PI questions: ***Create questions based around what you want students to be thinking and analyzing*** - if possible in ways that elucidate the process. For once in computing, getting the right output is of notably lesser importance.

8. CONCLUSIONS

This work seeks to encourage the adoption of PI in computer science classrooms by providing experience reports from PI instructors. Instructors reflect on the value of PI to monitor student understanding, engage them meaningfully with deep course concepts, and promote development of appropriate argument skills. We propose that widespread utilization of PI across the computing curriculum provides a fantastic opportunity to explicitly support student training in analysis, argumentation, communication, and teamwork skills.

9. ACKNOWLEDGEMENTS

This work was supported in part by the UK HEA-ICS.

10. REFERENCES

[1] Beatty, I. D., Gerace, W. J., Leonard, W.J., and Dufresne, R. J. Designing effective questions for classroom response system teaching. American Journal of Physics 74, 2006.

[2] Caldwell, J. E. Clickers in the large classroom: Current research and best-practice tips. CBE-Life Sciences Education 6, 2007.

[3] Carter, P. An experiment with online instruction and active learning in an introductory computing course for engineers: JiTT meets CS. 14th Western Canadian Conference on Computing Education, 2009.

[4] Crouch, C. H., and Mazur, E. Peer instruction: Ten years of experience and results. American Journal of Physics 69, 2001.

[5] Cutts, Q., Carbone, A., and van Haaster, K. Using an Electronic Voting System to Promote Active Reflection on Coursework Feedback. In Proceedings of Intl. Conf. on Computers in Education, Melbourne, Australia, 2004.

[6] Hake, R. R. Interactive-engagement vs. traditional methods: A six-thousand-student survey of mechanics test data for introductory physics courses. American Journal of Physics 66 (1), 1998.

[7] Knight, J. K. and Wood, W. B. Teaching more by lecturing less. Cell Biology Education 4, 2005.

[8] Ma, L., Ferguson, J., Roper, M., and Wood, M. Investigating the viability of mental models held by novice programmers. In Proceedings of the 38th SIGCSE technical symposium on computer science education, 2007.

[9] Pargas, R. P. and Shah, D. M. Things are clicking in computer science courses. In Proceedings of the 37th SIGCSE technical symposium on Computer science education, 2006.

[10] Simon, B., Kohanfars, M., Lee, J, Tamayo, K., and Cutts, Q. Experience report: Peer instruction in introductory computing. In Proceedings of the 41st SIGCSE technical symposium on computer science education, 2010.

[11] Smith, M., Wood, W., Adams, W., Wieman, C., Knight, J., Guild, N., Su, T. Why Peer Discussion Improves Student Performance on In-Class Concept Questions. Science 323, 2009.

[12] Wieman, C. and the staff of the CU and UBC Science Education Initiatives. Clicker Resource Guide, http://cwsei.ubc.ca.

[13] Zingaro, D. Experience report: Peer instruction in remedial computer science. In Proceedings of the 22nd World Conference on Educational Multimedia, Hypermedia & Telecommunications, 2010.