Facilitating Code-Writing in PI Classes

Daniel Zingaro
Ontario Institute for Studies in Education
University of Toronto
daniel.zingaro@utoronto.ca

Yuliya Cherenkova, Olessia Karpova,
and Andrew Petersen
University of Toronto Mississauga
{yuliya.cherenkova, olessia.karpova,
andrew.petersen}@utoronto.ca

ABSTRACT

We present the Python Classroom Response System, a web-based tool that enables instructors to use code-writing and multiple choice questions in a classroom setting. The system is designed to extend the principles of peer instruction, an active learning technique built around discussion of multiple-choice questions, into the domain of introductory programming education. Code submissions are evaluated by a suite of tests designed to highlight common misconceptions, so the instructor receives real-time feedback as students submit code. The system also allows an instructor to pull specific submissions into an editor and visualizer for use as in-class examples. We motivate the use of this system, describe its support for and extension of peer instruction, and offer use cases and scenarios for classroom implementation.

Categories and Subject Descriptors

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

Keywords

CS1, pair programming, peer investigation, active learning

1. INTRODUCTION

Peer Instruction (PI) is a pedagogical technique developed by the physics education community that has recently been applied to the teaching of computing courses. Physics educators realized that standard lectures are ineffective for teaching core concepts and addressing misconceptions [2], so PI was developed to introduce active, peer-based interaction to the classroom and to focus student attention on key misconceptions. Multiple research projects described in physics education journals demonstrate that PI-students outperform lecture-students on various grade-based and affective measures. For example, PI advantages have been demonstrated on final exams and standard concept inventories [2]. PI also tends to reduce student attrition and improve problem-solving ability [9].

Recent research in computing education suggests that PI may also be valuable as a pedagogical approach to teaching computing. Its focus on conceptual knowledge is appealing to computing educators who seek to confront misconceptions and help students understand core CS material [15]. However, PI focuses on in-class discussions of multiple-choice questions, and such a focus suggests a mismatch between PI and the skill-based practice required of CS1 students. Many of the skills typically expected of CS1 students relate to code-investigation and code-writing [5], and code-writing is disproportionately used to gauge student skill on final exams [12].

In this paper, we introduce a new tool, the Python Classroom Response System (PCRS), that can be used by computing teachers in classroom settings to administer both PI (multiple choice) and code-writing questions. Our goal is to enable PI educators to additionally ask code-writing questions and hence further adapt PI to our disciplinary context.

2. BACKGROUND

A PI course can be described by the pre-class, in-class, and post-class features that differentiate it from a lecture offering. Prior to each lecture, students are expected to read a section of the textbook and complete an associated reading quiz. We then replace the lecture with several iterations of the following four-step loop body:

Presage. Each PI iteration begins with a mini-lecture: a couple of slides that act as a refresher for what students read before class. The mini-lecture lasts at most a few minutes.

Engage. We next pose a ConcepTest [2] — a multiple choice question designed both to focus attention on and raise awareness of the key course concept from the mini-lecture [1]. After individually thinking about and voting on the correct answer (the solo vote), students discuss the question in small groups, reach a consensus, and vote again (the group vote). Students are encouraged to discuss each answer, verbalizing why it is correct or incorrect [15].

Gauge. While student voting data can be estimated using flashcards [8], the effectiveness of the "gauge" step is substantially enhanced by using electronic response systems. Clickers are a popular choice, as they provide immediate, accurate vote counts to the instructor and

SIGCSE'13, March 6–9, 2013, Denver, Colorado, USA.

compelling graphical displays for the students. Ideally, each response option for a ConcepTest will correspond to a common student misconception, so that the instructor can identify the range of understandings present among the students [4]. Seeing the histogram of results, students come to realize that they are not alone in their confusion [7], which may make them more comfortable voicing their concerns in subsequent, class-wide discussion.

Age. Having engaged students in collaborative debate and gauged their progress and understanding, the instructor leads a class discussion meant to solidify a mature conception of the lessons learned. The results of this discussion can be used by the instructor to adapt the lecture in real-time, filling gaps in student knowledge before progressing to the next topic and ConcepTest.

Using PI in combination with clickers provides an important pedagogical benefit to instructors in terms of response data. Following the lecture, the teacher can use this data to shape upcoming class meetings, as analyzing the responses to PI questions may expose points of confusion or difficulty that would benefit from extra class time.

Simon et al.'s paper at SIGCSE 2010 [15] began the investigation into the applicability and effectiveness of PI in CS. This original paper was primarily interested in the following question: do more students answer correctly after a group discussion compared to before the group discussion? To quantify such gains, the authors computed normalized gain (NG) on each question. NG captures the proportion of students who answer incorrectly in the solo vote but correctly in the group vote. For example, if 60% of students answer correctly in the solo vote and 80% answer correctly in the group vote, the NG for the question is 50% (i.e. 50% of "potential learners" demonstrated new understanding). The authors found an NG of 41% in a CS1 course and 35% in a CS1.5 course. Similarly, [17] used PI in a remedial CS1 course and found an NG of 29%.

End-of-course surveys consistently find that students and instructors value PI. A survey reported in [17] found, for example, that 100% of students believed that individually thinking about and responding to a PI question helped them learn the course material. In addition, 100% agreed that group discussion was helpful, and 94% strongly agreed that PI should be used in other courses. Other surveys corroborate these findings [11, 15]. A unique reflection of four instructors' use of PI [13] suggests other observed benefits, including a focus on teamwork and communication; flexible, demand-driven lectures; and opportunities for improving proof-reasoning abilities.

What kinds of questions can be asked using PI? Some of the papers cited above [15, 17] contain examples of questions where students must read and understand code. It is also possible to approach code-writing through the use of Parsons puzzles [17] that involve the rearrangement of existing code. In an analysis of a semester's worth of PI questions [3], three levels of abstraction were prominent: English, CS-speak, and code. Most prominently, questions often transitioned from one category to another; for example, a student might be asked for the code-equivalent of an English statement. Other questions were restricted to only one level of abstraction. Within coding, for example, many questions asked students to trace code to determine the code's output or purpose. However, despite this focus on code, multiple-choice questions inherently cannot require students to generate new code from scratch.

We acknowledge PI's utility in targeting various sought-after skills, such as code-reading, explaining code in English, and translating between levels of abstraction. However, we seek to extend the reach of PI into the domain of code-writing. We believe that having students write code, even small portions of code, is crucially important for success in CS1. Literature shows that code-writing is not equivalent to other code-related tasks such as code-reading or explaining code in plain English. Code-writing is argued to be at the top of a hierarchy with other code-related tasks below [16]. Furthermore, Zingaro et al. examined integrative code-writing questions and a corresponding set of "concept" questions on a final exam. Concept questions are small code-writing questions developed to target single concepts rather than multiple concepts, as is typical of code-writing questions. They found that concept questions were highly predictive of performance on the exam as a whole, suggesting that the ability to write small pieces of code from scratch is a valued outcome in CS1 [18]. This further suggests that code-based ConcepTests, which also focus on a small number of expected misconceptions, may also be effective performance evaluators. Validating this claim is the subject of ongoing work.

3. USE CASES

The design of the Python Classroom Response System (PCRS) emerges directly from our experience with PI pedagogy. Our goal was to extend PI to deal with discipline-specific needs while remaining faithful to the PI philosophy and 4-step cycle. As such, it was important not to lose the PI-support functionality that was previously available through the use of clickers. Therefore, assuming the availability of a wireless network and student handheld devices (e.g., laptop, phone, or tablet), the PCRS's multiple choice question support replaces the functionality of clicker software. Of course, clickers are low-cost devices much more affordable than the devices supported by the PCRS, but we anticipate that many students will already own a phone or other mobile device that will suffice.

Beyond traditional PI support, the PCRS can be used to support lecture-based code reading and code writing exercises. For example, we can use the PCRS to support code-writing exercises. In the past, we would insert programming exercises into lecture by asking students to discuss design issues in small groups and then to write pseudocode on paper. Later, we would implement one possible solution to a problem. However, during these exercises, students lacked many typical debugging resources, and in large classes, the instructor lacked feedback, as it is not feasible to review many students' code. Therefore, we were often unaware of the number of students answering correctly or exactly where students became stuck. The PCRS makes these exercises more effective by providing students and instructors with the support they require: students work interactively with a dynamic visualization tool that supports debugging and program understanding [14, 10], and instructors receive real-time feedback on their students progress.

The PCRS can also support a range of other code-writing exercises. For example, it supports partial-code exercises where students are provided with starter code that they

must read and then extend. In this case, the debugging tools aid the students in the code understanding task. In terms of debugging, the PCRS can also support defect triage exercises that are otherwise difficult to conduct in lecture. In this activity, we present students with incorrect or incomplete code that may pass some (but not all) test cases. We then ask students to determine why the specific tests fail and to correct the code using test-driven debugging techniques.

4. TOOL DESCRIPTION

The PCRS is a freely available, open-source system designed to support PI-style multiple choice questions as well as submission and analysis of live coding exercises in a classroom system (*link omitted*). It consists of administrator and student modules backed by a shared database. The administrator module contains functionality for creating questions and analyzing submissions in real-time or outside of class. The student module allows students to view questions, submit answers written in Python, and debug submissions to programming problems. The student module is built around the Online Python Tutor (OPT) developed by Philip Guo [6]. As shown in Figure 3, the OPT supports tracing and debugging activities by providing a visualization of the Python memory model and allowing students to step both forward and backward through the code.

The PCRS is entirely web-based, so it can be accessed through a browser on a laptop or tablet. Submitted code is executed on a server which returns an execution log to be visualized on the client machine. For security reasons, the code that can be executed and visualized is restricted: several built-in Python functions are disabled, and external modules cannot be imported. With the exception of file operations, which cannot be executed, the subset of Python that is supported is sufficient for exercises in a CS1 or CS2 course.

"L1 = [1, 2, 3]
L2 = L1
L2.append(4)
L2 = [0, 1, 2, 3]

After the preceding code has executed, what is the value associated with the variable L1?"

A. [0, 1, 2, 3]
B. [1, 2, 3]
C. [0, 1, 2, 3, 4]
D. [1, 2, 3, 4]

Figure 1: An example ConcepTest-style multiple choice question.

4.1 Multiple Choice Questions

To directly support PI, the PCRS allows instructors to define ConcepTest-style multiple choice questions like the one in Figure 1. Before class, the instructor enters the question and defines the possible answers. During class, the instructor logs into the administrator module of the PCRS and "enables" the problem when she wants students to respond to the question. At that point, it becomes visible to students who are logged into the PRS website, and they can submit responses. The instructor can display a response histogram in real-time and can "disable" the problem to stop submissions. The histogram displayed is very similar to the one displayed for programming questions illustrated by Figure 4.

"Given a number n and a Python list L, return the index (position) of the first occurrence of n in the list or -1 if n is not in the list."
1. n = 0, L = []
2. n = 0, L = [1, 2, 3]
3. n = 0, L = [1, 2, 0, 3]
4. n = 0, L = [1, 2, 3, 0]
5. n = 0, L = [1, 2, 0, 0, 3]

Figure 2: An example coding problem with the test cases defined by the instructor.

In most cases, PI questions are carefully crafted to uncover specific issues that the instructor anticipates will be misunderstood by students. For example, the question in Figure 1 evaluates whether students understand how assignment of mutable objects works in Python. Option (A) aligns with a common misunderstanding related to assigning variables (rather than the referenced object) and option (B) uncovers a misunderstanding about aliasing.

However, PI instructors often feel the need to quickly poll the students on unanticipated topics or questions raised during the flow of the class. The PCRS also provides instructors with the opportunity to ask a "snap question". From the administrator interface, a single click creates a new, enabled multiple choice question with 5 possible responses. The instructor has the choice of filling in text for the question or any of the responses, but none are required.

4.2 Programming Problems

The PCRS also allows programming instructors to ask PI-inspired programming questions. Like PCRS support for multiple choice questions, the instructor will usually define the problem before class and use real-time analysis functionality to identify misconceptions that are present. To create a coding question, the instructor must formulate the problem and, as in PI, specify a set of expected misconceptions. However, instead of encoding the misconceptions as multiple choice answers, the instructor implements them as a set of test cases. Each student submission is run through the test suite, and as illustrated in Figure 3, the instructor can identify the specific test cases that are proving troublesome for the class. Since each test case represents a typical misconception, the instructor can immediately identify the concepts on which to focus for the next part of the lecture.

Consider the code problem and corresponding test cases in Figure 2. In our experience, students often have difficulty solving it using a conditional ("while") loop. When we build the problem, we consider what problems students might encounter, and we build a test case for each one. We anticipate that students will attempt to access elements before checking the list's length (test case 1), continue iteration beyond the end of the list (case 2), terminate one comparison too early (case 4), or return the later index in cases with duplicate entries (case 5).

Before class, we quickly review the problem: test cases 1, 2, and 4 are related to understanding of list indices, and cases 3 and 5 check for correctness related to the problem

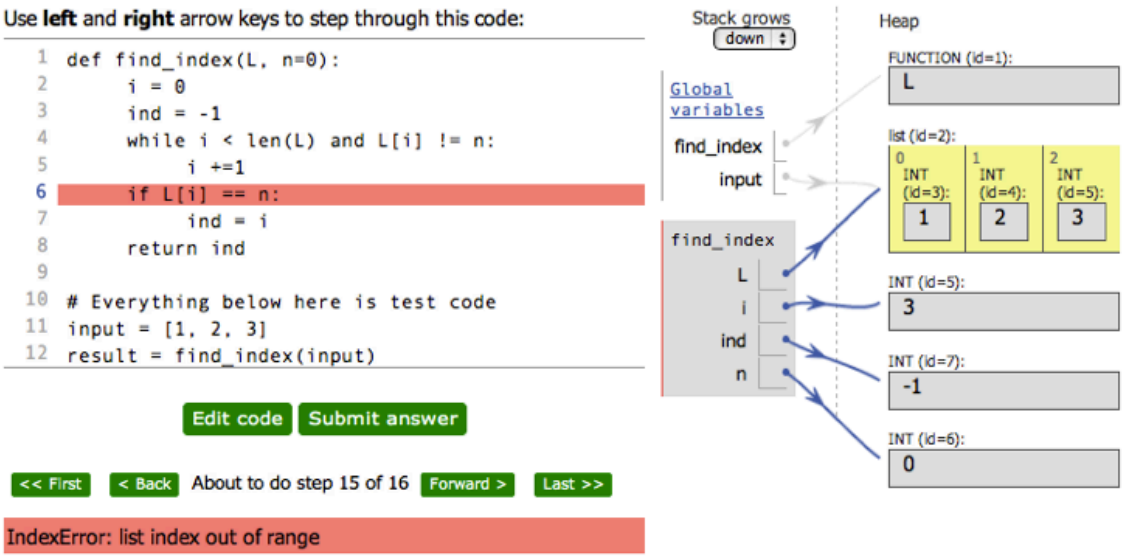

Figure 3: Encountering an error in Python code that iterates over a list.

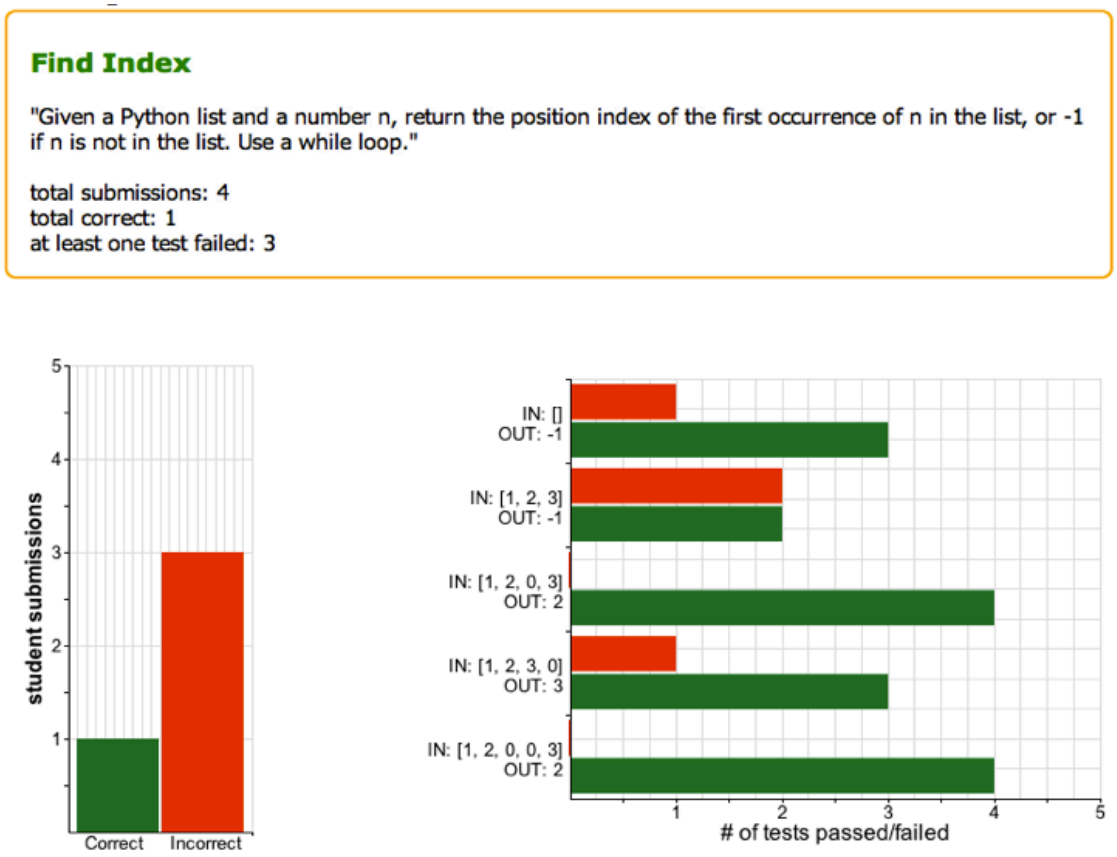

Figure 4: The instructor's view: monitoring submissions to a programming exercise. The graph on the left illustrates the number of submissions that passed all tests, while the graph on the right identifies the success and failure rate on particular tests.

specification. Then, after the problem is introduced, we can use real-time histograms produced by the PCRS to quickly identify the issues that are causing the students the most trouble. In response, we can issue verbal hints or pause the class and use a student submission to demonstrate why a particular test case is failing.

In the meantime, students are receiving feedback on their submissions directly from the system. Each submission is evaluated against the test cases, and the students are told which test cases pass (and fail). If they wish, they can use the visualizer to trace through a failing (or passing) test case, as demonstrated in Figure 3. In that example, the bottom two lines of code (11 and 12) tell the student exactly what test case was run; the top 8 lines are the student's submission. If the instructor desires, she can turn off this feedback feature. In that case, the students are told the number of test cases that fail but are not provided information about the contents of the test case. This might be desirable if an instructor uses the PCRS for marked work.

4.3 Analyzing Submissions

The PCRS provides several real-time and offline tools for analyzing student submissions. The real-time tools keep submissions anonymous, to protect student privacy, but the offline tools generally allow the instructor to identify the student in order to enable marking or credit for participation.

In real-time, the instructor can display a histogram of submissions. By default, each student's most recent submission is counted in the histogram, but all submissions are stored. To follow-up on a misconception, the instructor has access to a rich set of tools. The PCRS allows the instructor to browse submissions that pass (or fail) specific test cases and to display, modify, and trace through selected submissions in the same visualizer environment that the students use. This allows the instructor to find relevant examples submitted by students in the class and to use them in demonstrations. The PCRS also allows the instructor to filter submissions using a regular expression; this allows the instructor to identify submissions that use a specific syntactic structure.

Offline, the instructor can view student-specific information. On request, the PCRS can generate a comma-separated-value file that provides data for each student in the system. The instructor may specify which problems are included in the report; for the problems that are included, the correct (and incorrect) submission counts and pass/fail information for each test is provided.

The students are also provided facilities for analyzing their submissions. Each submission is stored in the database, and students can browse their history and send specific submissions to the visualizer for further investigation. Each submission is annotated with information on the tests that were passed and failed.

5. DISCUSSION

We believe that the PCRS is an effective means of implementing active, PI-inspired activities in a programming classroom. The tool is designed to give the students the opportunity to practice key code writing skills — problem solving, code production, and debugging — in class. At the same time, the instructor is obtaining rapid and rich feedback from the students and has the tools necessary to identify and use relevant student submissions in examples.

However, deploying this system requires planning. While most students have phones that can load the PCRS website, mobile phones do not have sufficient screen real estate to make visualization effective, and beyond that, coding on a phone keyboard is not feasible. While phones can easily be used for multiple-choice questions, we recommend that programming activities be performed on devices, like tablets or laptops, with larger screens and better text input functionality. In our own classes, only about 1 of 3 students normally bring a laptop to class, but we believe that when this tool is integrated into a class and students see its benefit, more will bring appropriate devices to lecture. Furthermore, since we prefer to structure the activities using a pair-programming or small-group model to get students involved in discussions, only 1/3 to 1/2 of the students need to be equipped with mobile devices.

Network resources are another issue that requires forethought. The local wireless point can become overwhelmed by connections if everyone in the classroom is using a device, so the network administrators need to be involved in the deployment of the PCRS in a large course.

Work on the PCRS is progressing in three directions. First, the interface and the analysis features of the tool are being improved based on instructor and student feedback. Support is also envisioned for languages other than Python. Second, we are performing a qualitative study investigating how the availability of tools like the PCRS change how students interact with the material, their peers, and their instructor. Third, we are preparing a set of pre-packaged programming exercises and multiple-choice questions to accompany the PCRS, so that new adopters can quickly deploy the tool.

6. CONCLUSION

While PI has been adopted in computing education contexts, such adoptions have tended to ignore the differences between computing and physics. In particular, recognition of the importance of code-writing exercises in CS1 as a key activity, assessment technique, and outcome measure has been lacking. We suggest that an active learning pedagogy used in a programming context should naturally support code-writing in addition to problem-solving, code-reading, and other valued CS1 skills. Our PCRS efforts seek to adapt PI to this unique disciplinary context to increase the breadth of questions and feedback that move between teacher and student during lectures. We make this tool freely available to the community, will continue development, and will report on its use by teachers that have previously used and not used PI.

7. REFERENCES

[1] I. D. Beatty, W. J. Gerace, W. J. Leonard, and R. J. Dufresne. Designing effective questions for classroom response system teaching. *American Journal of Physics*, 74:31–39, 2006.

[2] C. H. Crouch, J. Watkins, A. P. Fagen, and E. Mazur. Peer instruction: Engaging students one-on-one, all at once. In E. F. Redish and P. J. Cooney, editors, *Research-Based Reform of University Physics*. American Association of Physics Teachers, 2007.

[3] Q. Cutts, S. Esper, M. Fecho, S. R. Foster, and B. Simon. The abstraction transition taxonomy: Developing desired learning outcomes through the lens

of situated cognition. In *Proceedings of the Eighth International Workshop on Computing Education Research*, New York, NY, 2012.

[4] Q. Cutts, G. Kennedy, C. Mitchell, and S. Draper. Maximising dialogue in lectures using group response systems. 7th IASTED International Conference on Computers and Advanced Technology in Education. `www.dcs.gla.ac.uk/~quintin/papers/cate2004.pdf` (accessed August 19, 2011), 2004.

[5] U. Fuller, C. G. Johnson, T. Ahoniemi, D. Cukierman, I. Hernán-Losada, J. Jackova, E. Lahtinen, T. L. Lewis, D. M. Thompson, C. Riedesel, and E. Thompson. Developing a computer science-specific learning taxonomy. *SIGCSE Bulletin*, 39(4):152–170, Dec. 2007.

[6] P. J. Guo. Online Python Tutor: Embeddable web-based program visualization for CS education. In *Proceedings of the 44th SIGCSE Technical Symposium on Computer Science Education*, 2013.

[7] J. K. Knight and W. B. Wood. Teaching more by lecturing less. *Cell Biology Education*, 4:298–310, 2005.

[8] N. Lasry. Clickers or flashcards: Is there really a difference? *The Physics Teacher*, 46:242–244, 2008.

[9] N. Lasry, E. Mazur, and J. Watkins. Peer instruction: From harvard to the two-year college. *American Journal of Physics*, 76:1066–1069, 2008.

[10] T. L. Naps, G. Rössling, V. Almstrum, W. Dann, R. Fleischer, C. Hundhausen, A. Korhonen, L. Malmi, M. McNally, S. Rodger, and J. A. Velázquez-Iturbide. Exploring the role of visualization and engagement in computer science education. *SIGCSE Bulletin*, 35(2):131–152, June 2002.

[11] R. P. Pargas and D. M. Shah. Things are clicking in computer science courses. In *Proceedings of the 37th SIGCSE Technical Symposium on Computer Science Education*, pages 474–478, New York, NY, 2006.

[12] A. Petersen, M. Craig, and D. Zingaro. Reviewing CS1 exam question content. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education*, pages 631–636, New York, NY, 2011.

[13] L. Porter, C. Bailey Lee, B. Simon, Q. Cutts, and D. Zingaro. Experience report: a multi-classroom report on the value of peer instruction. In *Proceedings of the 16th Annual Joint Conference on Innovation and Technology in Computer Science Education*, pages 138–142, New York, NY, 2011.

[14] N. Ragonis and M. Ben-Ari. On understanding the statics and dynamics of object-oriented programs. In *Proceedings of the 36th SIGCSE Technical Symposium on Computer Science Education*, pages 226–230, New York, NY, 2005.

[15] B. Simon, M. Kohanfars, J. Lee, K. Tamayo, and Q. Cutts. Experience report: Peer instruction in introductory computing. In *Proceedings of the 41st SIGCSE Technical Symposium on Computer Science Education*, pages 341–345, New York, NY, 2010.

[16] J. L. Whalley, R. Lister, E. Thompson, T. Clear, P. Robbins, P. K. A. Kumar, and C. Prasad. An australasian study of reading and comprehension skills in novice programmers, using the bloom and solo taxonomies. In *Proceedings of the 8th Australasian Conference on Computing Education*, pages 243–252, Darlinghurst, Australia, 2006.

[17] D. Zingaro. Experience report: Peer instruction in remedial computer science. In *Proceedings of the 22nd World Conference on Educational Multimedia, Hypermedia & Telecommunications*, pages 5030–5035. AACE, 2010.

[18] D. Zingaro, A. Petersen, and M. Craig. Stepping up to integrative questions on CS1 exams. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, pages 253–258, New York, NY, 2012.