Evaluating a Breadth-First CS 1 for Scientists

Zachary Dodds, Ran Libeskind-Hadas, Christine Alvarado, Geoff Kuenning
Harvey Mudd College Computer Science Department
301 Platt Boulevard
Claremont, CA 91711
909-607-1813

{dodds, hadas, alvarado, geoff}@cs.hmc.edu

ABSTRACT

This paper presents a thorough evaluation of *CS for* Scientists, a CS 1 course designed to provide future scientists with an overview of the discipline. The course takes a breadth-first approach that leverages its students' interest and experience in science, mathematics, and engineering. In contrast to many other styles of CS 1, this course does not presume that its students will study more computer science, but it does seek to prepare them should they choose to. We summarize the past year's worth of assessments of student learning, retention, and affect – with particular attention paid to women's voices. Where possible, we contrast these student measures with those from a traditional, imperative-first CS1 that this new course replaced. The data thus far suggest that *CS for Scientists* significantly improves students' understanding of CS, its applications, and practice.

Categories and Subject Descriptors

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

General Terms

Measurement, Design, Human Factors

Keywords

CS for scientists, introductory CS, CS 1 assessment

1. CS FOR SCIENTISTS

Scrutiny seems an unavoidable fate for introductory computer science. In a field as dynamic as CS, we who teach CS0 and CS1 should strive to remain relevant and current. At the same time, we try to retain those topics and skills that enable our students to cope with next year's changes as well as last year's. This balance is particularly delicate when designing introductory CS for scientists. The evolving impact of CS on all scientific disciplines has been dramatic and well documented, e.g., [29][31]. As George Johnson put it, "All science is computer science." [17]

To leverage CS's growing importance, in 2006 we replaced our traditional, Java-based, objects-late CS 1 course with a Python-based, breadth-first course nicknamed *CS for Scientists* [1]. Our goal was to create a curriculum "suitable for any student intending to major in science or engineering (including CS students)." [26] In particular, we hoped this new offering would (1) develop programming and problem-solving skills useful across engineering, mathematics, and the natural sciences, (2) attract students to continue with CS, and (3) provide a coherent, intellectually compelling picture of CS, even as final CS course.

1.1 Context and Related Work

It is a wonderful time to teach CS 1! Curricular innovations within introductory CS are inspiring and numerous. Many of these experiments draw strength in a similar manner: by weaving a *thematic structure* amid introductory CS topics [15][24].

One of the most widespread of these themes for introductory CS is media computation [14][21]. Other themes now scaffolding CS1 include games [3][13][19][32], robotics [5], computer vision [22], and art [12][27]. In each of these cases, the thematic overlay tends to pull away from CS and toward the specifics of the course's theme. Throughout *CS for Scientists* we strove to keep the focus on CS, with applications motivating that focus.

Science and engineering enjoys a long history as a CS theme [2][16][20][28]. Yet these experiments, both new and old, tend to be service courses rather than CS *per se*, e.g., they do not contribute to a CS degree. Courses like [8] and [30] present facets of CS to specialists in other disciplines. Our course, on the other hand, represents a full-fledged CS 1 designed to generate interest in and prepare students for additional courses within the field.

Pedagogically popular styles of CS 1, such as imperative-first or objects-first [10], all make the implicit assumption that there will be something *second*. We knew that only a fraction of our students would continue with CS, though we hoped to make it a sizeable fraction! We hypothesized that breadth-first would best suit students for whom the class might also be *breadth-last*. Breadth-first CS is far from new. As CC2001 summarizes aptly,

> the breadth-first model has not enjoyed the success that its proponents had envisioned… most breadth-first courses that exist today seem to be lead-ins to a more traditional programming sequence. This model, which has several successful implementations, is outlined in CS100B… [10]

Our *CS for Scientists* takes this hard-won experience to heart; CS100B is our curriculum's basis. Yet our course has significant shifts in emphasis to serve future scientists and engineers: multi-paradigm programming, leveraging existing code, CS's influence on science today, and acknowledging the reality that many students would not be able to take another CS course afterwards.

Perhaps the work closest to *CS for Scientists* is Sedgewick and Wayne's forthcoming text [26] and the Princeton University course, COS126, from which it has grown. While our debt to COS126 is strong in spirit, several differences distinguish our offering. First, rather than work only within the object-oriented paradigm offered by Java, we take a multiparadigm approach via Python. Because of this change, we have contributed a freely-available body of Python software to support our assignments and laboratories. Further, we have been able to compare students' academic and affective changes across our old and new offerings. Here, we report the results of this year-long assessment effort.

1.2 Contributions

This paper presents three specific contributions that *CS for Scientists* adds to the efforts begun in [24][35]:

- ***Complete CS 1 materials.*** Two fifteen-week sets of labs, assignments, and lecture slides are freely available [1].
- ***Student-centric evaluation.*** We report results from pre- and post-term metrics of student engagement and understanding. We have followed retention trends and compared the impact of the new curriculum with the Java-based course it replaced.
- ***Tracking gender cohorts.*** Where possible, these assessments trace the impacts on women and men both together and separately.

The data suggest that CS for Scientists succeeds in three ways: students "get" the importance of CS to their future scientific endeavors and its importance as a stand-alone discipline; students succeed in future CS courses to a greater extent; and they continue studying CS as often – and perhaps more – than from traditional alternatives. In each measure women benefit at least as much as men. We hope that this curriculum, its support materials, and its assessments will be of service to other CS educators serving future scientists, mathematicians, and engineers.

2. CURRICULUM

186 first-year college students took *CS for Scientists* in the fall of 2006: all of them are pursuing a degree in some natural science, i.e., mathematics, engineering, physics, chemistry, biology, CS, or combinations of these fields. Though scientists for sure, the majority have not yet chosen their major field of study, a decision not required until the end of the second year at our institution.

To implement our breadth-first curriculum, we broke the semester into five three-week units (see Table 1) in which students would learn and practice different programming paradigms. In order to support all of these with a minimum of syntactic overhead, we heed [6] and [9] in choosing a multi-paradigm language, Python.

Table 1. Summary of *CS for Scientists*' Curriculum

Weeks	Paradigm	Samples of the labs and assignments
1-3	functional	integration, random walks, ciphers
4-6	low-level	recursion in assembly, 4-bit multiplier
7-9	imperative	Markov text generation, game of life
10-12	objects/classes	Connect Four player, sudoku solver
13-15	CS theory	uncomputability, finite-state machines

2.1 Modules, example labs, and assignments

To set the stage for the assessment results, we briefly summarize *CS for Scientists*'s content. Details appear in [1].

While software engineers will persuasively argue that objects and classes are computation's basic building blocks, practicing scientists and engineers often have smaller-scale needs, e.g., quick scripts to analyze, summarize, or reformat data. Our course thus begins with small, task-specific functions as the basic building blocks of computation. Turtle graphics, numerical integration, the Caesar cipher, random walks, and 1d cellular automata exemplify recursion and functional programming.

The second module, "low-level" computation, reinforces the idea of modularity and composition via logic gates: students built 4-bit ripple-carry adders from AND, NOT, and OR gates in Carl Burch's outstanding Logisim tool [7]. Those adders became building blocks in 4-bit multipliers. Combinational-circuit design segues to larger-scale computer architecture: students capped this experience by implementing recursion (the stack and function calls) in a Python-based assembly language simulator.

Register-level `jmp` and `cmp` assembly instructions transition naturally to the repetitive control structures and variable reassignment of imperative programming, our third module. Students implement the game of life, Gaussian elimination, and Flesch readability, among other assignments.

Students then augment procedural programming with class- and object-based constructs. They create a `Date` class to answer questions like "how many days apart are June 25, 2007 and February 30, 1712" and "which day of the week is most likely to be the 13th of the month."† Implementing Connect Four provides deeper design practice with both OOP and 2d arrays.

A medium-sized final project further exercises object-oriented style, with students choosing among three options: a physical simulator and GUI for a game of pool; a state-based controller navigating a simulated robot through a continuous environment; or a finite-automaton simulator, with graphics, of a small space-filling agent similar to Karel [4].

The finite-state machines used in these latter two final projects complement in-class exercises on (un)computability and deterministic finite automata. These lectures, reinforced by final exam questions, wrap up the term with a bird's-eye view of what computation can and can't do, e.g., the halting problem and several other uncomputable functions. Figure 1 shows a few examples of student work from fall 2006. On average, students completed four programming problems per week.

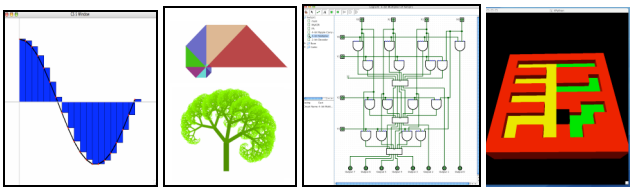

Figure 1. Student work from fall 2006's *CS for Scientists* ***L to R:*** visualizing numeric integration**,** turtle graphics, a four-bit multiplier circuit**,** a Karel-like automaton exploring its world.

† Sweden observed Feb. 30, 1712, which fell 107,852 days before June 25, 2007. Friday is strictly more common as the 13th than any other day of the week.

3. EXPERIENCES AND EVALUATION

Changing curriculum is one thing; changing students is another. With our redesign of introductory computer science, we hoped that students would (1) demonstrate an appreciation for the diversity of applications and investigations undertaken in CS, (2) learn a broader and more representative set of computer science topics, (3) exhibit the programming and computational-thinking skills required for success in CS 2, (4) choose to continue their study of CS in greater numbers, and (5) enjoy their *CS for Scientists* experience.

We have sought throughout to assess how women have experienced the course, as distinct from men. Increasing the number of women majors outranks growth per se as a priority for our department. We assessed student behaviors through content exams, opinion questionnaires, course evaluations, per-problem feedback and scores, enrollment numbers, and head-to-head comparisons with cohorts from the previous, imperative-first CS 1 curriculum. We next address the five objectives above, in turn.

3.1 Students' appreciation of CS's diversity

In order to evaluate how students' perceptions of computer science changed through the semester, we asked two questions on both the first and the last day of classes: "What is computer science?" and "Describe one thing a researcher in CS might study."

The responses to "What is CS?" have been coded into four levels of sophistication: **Level 1 (none)** represents non-answers such as "the science of computers" or "the study of technology," as well as purely derivative/analytic ones, e.g., "using code to get computers to do things" or "figuring out how computers work." Responses that articulate some of synthesis or breadth within CS are **Level 2 (naive)**, e.g., "software and hardware design" and "coding, debugging, and analyzing problems to develop computer-based solutions." Answers that acknowledged the field beyond physical computers and their software became **Level 3 (basics)**, e.g., "the study of computational algorithms and their applications." Finally, the most nuanced answers become **Level 4 (details)**: e.g., "a lot is about general, language-agnostic algorithms and relative merits of speed and efficiency - in some cases actually wondering how and if something is possible. CS is not programming, it is implemented in programming."

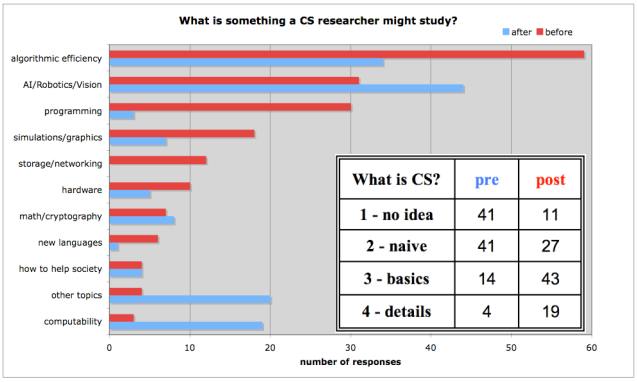

Figure 2. "What is CS?" student thoughts, before and after CS 1

For us these results drove home the unexamined preconceptions that many students bring to a first CS course. Particularly worrying, though perhaps unsurprising among a science-focused group, was the commonly held belief that CS is simply tech support for other disciplines. On the other hand, significantly more students (t=0.05) cited a computational application *per se* in the end-of-term survey; the same significant shift appeared in the more nuanced understanding of CS as an independent and compelling field of study. These results suggest that a science-themed curriculum need not relegate CS as servant to other fields. Rather, by "starting where the students are," the curriculum can leverage existing passions to reframe CS's many roles among mathematics, science, and engineering today.

Time-usage surveys reinforce this hypothesis, with CS motivating the greatest amount of work among first-year students' courses:

Core curriculum students' hours per week (and anticipated grades), Fall '06

	Physics	CS 1	Math	Writing	Bio	Engin.
Women	**5.2** (C+)	**7.2** (B)	**6.4** (B-)	**5.3** (B-)	**2.8** (B-)	**7.0** (B-)
Men	**5.9** (B-)	**7.3** (B+)	**6.6** (B)	**5.7** (B-)	**2.3** (B)	**6.2** (B)

Figure 3. Student-reported hours per week in the required first-year curriculum. Anticipated grades are in parentheses.

It is unlikely that Figure 3's data stem from anxiety about passing the course. On the same survey, students reported their expected grades in each class: both men and women anticipated higher grades in CS 1 than in their other classes. Yet these figures speak only to the new *CS for Scientists*; previous students did not take these surveys. The next two questions, on the other hand, directly contrast student learning between the old and new CS 1 curricula.

3.2 Student learning of CS topics and skills

The 97% passing rate for the new CS 1 suggests that the '06 students ***might*** have absorbed a larger, more representative cross-section of computer science topics. The new course did cover a strict superset of the topics in the Java-based, imperative-first CS 1 that it replaced. The final exam questions summarize this concisely in Figure 4. At right are the old vs. new median final exam scores, along with the median scores for women, men, and all students in three facets of the course.

Old and new exam questions

1. Create and compose three small functions to spec.
2. Demonstrate correct use of references vs. raw data
3. Manipulate a 1d data structure, e.g., `mode`
4. Write a program searching a 2d array along both dims.
5. Design/write a `Time` class with data, methods to spec.

Additional new exam questions

6. Design/sketch a novel digital circuit from its I/O spec.
7. Compose an assembly-language routine to spec.
8. Prove a (previously unseen) function uncomputable

Final exam median scores

New CS1	77%	Old CS1	85%

New CS 1	All	Women	Men
Final exam	76.9	71.5	79.0
Full course	89.9	89.6	90.5
Homework	92.3	92.8	92.3

Figure 4. Exam topics and performance, new vs. old CS 1

On first blush, the results might seem disheartening: the new final exam median of 78% is considerably less than the prior exam's median of 85%. What's more, differences in final exam scores between male in female students are significant (t=.05), with a gap of seven points. This might reflect the tendency for men to arrive at college with more CS experience than women. Indeed, women averaged higher on HW assignments; differences in overall grades were not significant.

The drop in exam scores from the previous CS 1 course also suggests a tradeoff: differences in breadth make it possible that *more* material was learned, despite the downward trend in scores. To test whether breadth really came at the expense of depth, we compared the student cohorts' CS 2 performances.

3.3 How did students do in CS 2 ?

Our spring 2007 CS 2 course remained unchanged from previous offerings. Half of CS 2 teaches Java, focusing on object-oriented design of list and tree data structures, along with graphics and event-based programming. The other half covers a variety of topics - graph algorithms, parsing, turing machines, regular languages, and logic programming - using Scheme, Prolog and JFLAP. No Python at all is used in CS 2. The spring 2007 CS 2 course comprised 32 students from the new, python-based CS 1 and 13 students from the older, java-based CS 1. Figure 5 reports midterm and final exam scores from these groups.

CS 2, Spring '07	Midterm exam scores, %			Final exam scores, %		
	women	men	overall	women	men	overall
Java CS 1	84.0 (1)	79.5 (12)	79.8 (13)	92.0 (1)	78.4 (9)	79.8 (10)
Python CS 1	80.7 (7)	84.7 (25)	83.8 (32)	88.1 (7)	84.3 (24)	85.1 (31)

Figure 5. Comparing old vs. new CS 1 students within a single CS 2 course. The size of each cohort appears in parentheses.

The data in Figure 5 confirm our suspicions that using python early does not disadvantage those students who pursue a typical CS major. The all-student differences in means in Figure 5 are not huge, but they are significant at the t=0.15 level. Given that python was *not* used at all in CS 2, and that CS 2's largest emphasis was Java and objects (as in the *old* CS 1), we feel that even the lack of statistically significant performance differences yields a pedagogically significant result.

That only one woman had returned from the previous CS 1 reveals a crucial impetus for our new *CS for Scientists* curriculum. That the women from the new CS 1 fared as well as – or better than – men extends the encouraging findings of [24] that a compelling thematic structure within CS 1 can serve *all* students well, even with a theme as broad as science and engineering.

To be sure, Figure 5's results showing equal (or better) CS 2 performance after the new CS 1 are meaningful only if students *in comparable numbers* continue to pursue CS beyond the first course. If major numbers drop, after all, it might be self-selection and not pedagogical effectiveness that explain an increase in knowledge and skill development.

3.4 Did students continue studying CS?

As of this writing, no students from the '06 offering of *CS for Scientists* have declared a major; that decision comes halfway through sophomore year. Yet we do have two semesters of CS 2 enrollments: Figure 6 tracks retention into CS 2 during first-year spring and sophomore fall from 2004 to 2007.

Figure 6. Comparing retention into CS 2 during the first two terms after CS 1. Only the rightmost data are for the new CS 1.

These retention numbers are among the most disappointing of this experiment: we had hoped to see significant increases in our CS 2 numbers. Because major choice is a zero-sum game, political considerations forbid interview or questionnaire investigations for the reasons behind Figure 6's flatlining. One possibility worth considering, however, is simultaneously the most humbling and heartening. Perhaps major choices are both too personal and too important to depend much on first impressions.

3.5 Affective evaluations

As the CS education community aptly points out, "my students liked it!" is more anecdote than evidence, particularly in questions of student learning [18][34]. Although we have sought to reach beyond opinion for metrics of student change, we also believe that affect matters. Majors and nonmajors alike will carry their feelings about CS wherever their future work takes them. Even students who decide that one term of CS is enough, we hope, will retain positive associations with the field.

Because this course replaced a procedural-then-objects Java course, we follow parallel evaluations' lead [11][33] in asking how student opinion differed between the old and new offerings. Figure 7 summarizes students' Likert responses from 1 (least agreement) to 7 (most agreement) with the statements *(A) The course stimulated my interest in the subject matter* and *(B) I learned a great deal in this course*. Students found *CS for Scientists* more compelling and informative both in absolute terms and relative to the balance of their course loads.

from 1-7	All F'05 courses	F'05 old CS 1	F'06 new CS 1	All F'06 courses
Question A	5.65	5.14	5.89	5.70
Question B	5.71	5.81	6.11	5.80

Figure 7. Comparison of student opinions of CS 1 before (2005) and after (2006) introducing the *CS for Scientists* curriculum.

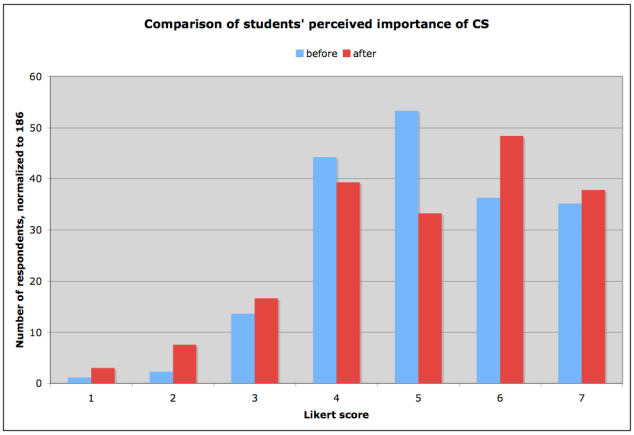

Figure 8. Students' perceptions of the importance of CS *to them personally*, measured before and after the new course.

We also measured 2006 students' perceptions of CS's impact on them personally by asking "how important do you think CS or programming skills will be in your future" with Likert-scale scores ranging from 1 – *not at all* to 7 – *very*. Figure 8 illustrates the results from before and after *CS for Scientists*. While the means of these before-and-after distributions are identical at 5.1, their shapes are not. Indeed, the heavier tails on both ends of the

latter scale suggest that more students have “taken sides” as to whether or not they feel comfortable and eager to build upon their CS skills in the future.

We believe the breadth-first approach explains both the improvements of Figure 7 and the allegiance-splitting apparent in Figure 8. Different students found themselves drawn to different pieces of the course: some enjoyed the mathematical and theoretical unit, others preferred the programming and problem-solving, still others gravitated to the circuit design and connections with electrical and computer engineering. The instructors noticed that the changes of pace offered psychological "breathers" for those students who struggled with the mechanics and semantics of python, even as coding remained the focus of 13 of the 15 weeks of the course. By the end of the experience, students felt that they had a strong foundation on which to judge their personal interest in the field.

4. VERDICT

From these data and in looking back broadly at our initial offering of *CS for Scientists*, we are optimistic about its approach to teaching future scientists introductory computer science. As always, there remain a number of rough edges that we look forward to addressing in subsequent offerings.

We recognize, too, that Albert Shankar's infamous dictum, "All educational experiments are doomed to succeed" plays an unavoidable role in these assessments. Only time will tell the extent to which these results are a by-product of novelty and enthusiasm or results of the structural changes undertaken. We will track student data through 2007 and 2008 in order to better understand the underlying factors at work. Similarly, we look forward to collaborating with other institutions to develop CS 1 curricula that are useful and inspiring for mathematicians, scientists, and engineers of all stripes.

5. REFERENCES

[1] CS 5 website, https://www.cs.hmc.edu/twiki/bin/view/CS5/WebHome

[2] Bachnak, R. and Steidley, C. An interdisciplinary laboratory for computer science and engineering technology. *Journal of Computing Sciences in Colleges* 17(5) April 2002, 186-192.

[3] Bayliss, J. D. and Strout, S. Games as a "flavor" of CS1. In *Proc. SIGCSE 2006; Houston, TX, USA*[A], 500-504.

[4] Bergin, J., Roberts, J., Pattis, R., and Stehlik, M. *Karel++: A Gentle Introduction to the Art of Object-Oriented Programming*. John Wiley & Sons, NY, NY, 1996

[5] Blank, D. Robots Make Computer Science Personal. *Communications of the ACM* 49(12) (Dec. 2006), 25-27.

[6] Budd, T. A. and Pandey, R. K. Never mind the paradigm, what about multiparadigm languages? *ACM SIGCSE Bulletin* 27(2) (June 1995), 25-30.

[7] Burch, C. Logisim: a graphical system for logic circuit design and simulation. *Journal on Ed. Resources in Computing (JERIC)* [A] 2(1) (3/2002), 5-16.

[8] Burhans, D. T. and Skuse, G. R. The role of computer science in undergraduate bioinformatics education. In *Proc. SIGCSE 2004; Norfolk, VA, USA*[A], 417-421.

[9] Close, R., Kopec, D., and Aman, J. CS1: perspectives on programming languages and the breadth-first approach. In *Proc. CCSCNE 2000; Mahwah, NJ, USA*[A], 228-234.

[10] Computing Curricula 2001. *Journal on Educational Resources in Computing*[A], Joint Task Force on Computing Curricula, eds. Volume 1, Issue 3es.

[11] Crescenzi, P., Loreti, M. and Pugliese, R. Assessing CS1 java skills: a three-year experience. In *Proc. ITiCSE 2006; Bologna, Italy*[A], 348.

[12] Davis, T. A. and Kundert-Gibbs, J. The role of computer science in digital production arts. In *Proc. ITiCSE 2006; Bologna, Italy*[A], 73-77.

[13] Giguette, R. The Crawfish and the Aztec treasure maze: adventures in data structures. *ACM SIGCSE Bulletin* 34(4) (Dec. 2002), 89-93.

[14] Guzdial, M. A media computation course for non-majors. In *Proc. ITiCSE '03; Thessaloniki, Greece*[A], 104-108.

[15] Guzdial, M. and Tew, A. E. Imagineering inauthentic legitimate peripheral participation: an instructional design approach for motivating computing education. In *Proc. ICER 2006; Canterbury, UK*[A], 51-58.

[16] Jehn, L. A., Rine, D. C., and Sondak, N. Computer science and engineering education: Current trends, new dimensions and related professional programs. In *Proc. SIGCSE 1978; Pittsburgh, PA, USA*[A], 162-178.

[17] Johnson, George. All Science is Computer Science. *New York Times* March 25, 2001.

[18] Kinnunen, P., McCartney, R., Murphy, L., and Thomas, L. *Through the eyes of instructors: A phenomenographic investigation of student success.* In the proceedings of ICER, September 14-15, 2007, Atlanta, GA, USA[A]

[19] Ladd, B. C. The curse of Monkey Island: holding the attention of students weaned on computer games. *Journal of Computing Sciences in Colleges* 21(6) (June 2006), 162-174.

[20] Lambrix, P. and Kamkar, M. Computer science as an integrated part of engineering education. In *Proc. ITICSE 1998; Dublin, Ireland*[A], 153-156.

[21] Matzko, S. and Davis, T. A. Teaching CS1 with graphics and C. In *Proc. ITiCSE 2006; Bologna, Italy*[A], 168-172.

[22] Olson, C. F. Encouraging the development of undergraduate researchers in computer vision. In *Proc. ITiCSE 2006; Bologna, Italy*[A], 255-259.

[23] Paul, J. Leveraging students' knowledge: introducing CS 1 concepts. *Journal of Computing Sciences in Colleges* 22(1) (Oct. 2006), 246-252.

[24] Rich, L., Perry, H., and Guzdial, M. *A CS1 course designed to address interests of women*. In *Proc. 35th SIGCSE '04*, March 3-7, Norfolk, Virginia, USA[A],190-194.

[25] Scherer, D., Dubois, P., and Sherwood, B. VPython: 3D interactive scientific graphics for students. *Computing in Science and Eng.* 2(5) 2000, 56-62.

[26] Sedgewick, R. and Wayne, K. *Introduction to Programming (in Java)*, preliminary version, Pearson Addison Wesley, 2006. ISBN 0-536-31807-7.

[27] Smith King, L. A. and Barr, J. Computer science for the artist. In *Proc. SIGCSE 1997; San Jose, CA, USA*[A], 150-153.

[28] Stevenson, D. E. Science, computational science, and computer science: at a crossroads. In *Proc. ACM '93; Indianapolis, IN, USA*[A], 7-14.

[29] Steering the future of Computing. *Nature* 440(7083) (March 2006 special issue on 2020 Computing), 383-580.

[30] Tesser, H., Al-Haddad, H. and Anderson, G. Instrumentation: a multi-science integrated sequence. In *Proc. SIGCSE 2000; Austin, TX, USA*[A], 232-236.

[31] Towards 2020 Science, by the 2020 Science Expert Group. Microsoft Press, Redmond, WA, USA. 2006.

[32] Wallace, S. A. and Nierman, A. Addressing the need for a java based game curriculum. *Journal of Computing Sciences in Colleges* 22(2) 12/2006, 20-26.

[33] Weir, G. R. S., Vilner, T., Mendes, A. J., and Nordström, M. Difficulties teaching Java in CS1 and how we aim to solve them. In *Proc. ITiCSE '05; Caparica, Portugal*[A], 344-345.

[34] Winters, T. and Payne, T. *What do students know? An outcomes-based assessment system*. In *Proc. ICER 2005*, Oct. 1-2, Seattle, WA, USA[A], 165–172.

[35] Zelle, J. *Python Programming: An Introduction to Computer Science*. Franklin, Beedle & Associates. Wilsonville, OR. 2004. ISBN 1-887902-99-6.

[A] ACM Press, New York, New York, USA