# Experience Report:
# CS1 for Majors with Media Computation

Beth Simon, Päivi Kinnunen, Leo Porter
Computer Science and Engr. Dept
University of California, San Diego
La Jolla, CA 92093-0404
+01 858 534 5419
{bsimon, pkinnunen, leporter}@cs.ucsd.edu

Dov Zazkis
Math and Science Education
San Diego State University/
University of California, San Diego
San Diego, CA 92120
+01 619 594 4696
dzazkis@ucsd.edu

## ABSTRACT
Previous reports of a media computation approach to teaching programming have either focused on pre-CS1 courses or courses for non-majors. We report the adoption of a media computation context in a majors' CS1 course at a large, selective R1 institution in the U.S. The main goal was to increase retention of majors, but do so by replacing the traditional CS1 course directly (fully preparing students for the subsequent course). In this paper we provide an experience report for instructors interested in this approach. We compare a traditional CS1 with a media computation CS1 in terms of desired student competencies (analyzed via programming assignments and exams) and find the media computation approach to focus more on problem solving and less on language issues. In comparing student success (analyzed via pass rates and retention rates one year later) we find pass rates to be statistically significantly higher with media computation both for majors and for the class as a whole. We give examples of media computation exam questions and programming assignments and share student and instructor experiences including advice for the new instructor.

## Categories and Subject Descriptors
K.3.2 [**Computer Science Education**]: Introductory Programming – *abstract programming concepts*

## General Terms
Human Factors, Languages.

## Keywords
CS1, Media computation, Retention.

## 1. INTRODUCTION
The use of a contextualized approach to CS1, through media computation has been shown to increase student retention in introductory programming courses [2,3,7,9]. The most common approach follows a textbook (in python or Java) from Guzdial and Ericson [4] and has been tracked at Georgia Tech since 2003 in an introductory course for non-majors. To our knowledge, no study has yet reported on the use of such an approach in the standard first course for computing majors (CS1).



Given high attrition rates in CS1 courses (15-20%, locally), an approach that improves retention rates would be valued. But would retention come at a cost? Can it be done without sacrificing desired concept or Java construct competencies? Would students really learn "the same things" they did previously, or would it require material to be pushed into a later course?

We report on the implementation of a media computation-based CS1 course at a large, selective R1 institution in the U.S and its impact on computing majors. We seek to provide information and comparisons to help guide instructors interested in adopting media computation for a majors' course. We compare one year's courses in the traditional approach with one year's courses using media computation – all taught by the same instructor (not a developer of the media computation approach).

We use post hoc analysis to consider whether students in the media computation approach received the same experience with standard programming concepts (for loops, if-statement, objects etc.) and their implementation/use in Java. This is accomplished through comparison of exam questions and programming assignments. Additionally, we report the success of majors though both course pass rates and retention in the major a year later. We also report general course pass rates for comparison with previous work. Finally, we discuss both positive and challenging aspects of the media computation approach as experienced by the instructor and the students.

## 2. RELATED WORK
Media computation as a context for introductory programming was originally designed to serve non-CS majors with a variety of interests and backgrounds. Its design addresses three aspects found to be barriers to students' success in computer science: relevancy, creativity, and social aspect of learning. These issues are regarded to be especially relevant for females [6] and non-CS major students [9]. The digital media-based approach introduces programming in a context for which students commonly (and enjoyably) already find computers useful. Open-ended assignments supported by on-line forums (to share end products) provide a creative outlet and counteract the stereotypes of computing as boring and asocial [9].

[2,3,7,8] have reported lower DWF (fail) rates in various media computation courses compared to traditional CS1 courses. However, all of these courses were either for non-majors or designed as a pre-majors course (CS0.5). Additionally, [3,8,9] report an increased interest among non-CS majors towards taking additional media computation courses.

This work adds to the existing related work by reporting on the use of a media computation course designed as a CS1 for majors. Also, beyond commonly reported retention rates, we analyze the expected competencies of students in the course.

# 3. BACKGROUND

In this study we compare four 10-week courses all taught by the same instructor at the same institution. Two courses (Fall 2006, Winter 2007) were taught with a traditional objects-early approach with a popular textbook in Java. The other two courses (Winter 2009, Fall 2009) were taught using a media computation approach supported by the Java version of the Guzdial and Ericson textbook. A prototype media computation course was taught in Fall 2008, but is not reported here since significant development in the course has occurred since.

The motivation for switching the introductory course for majors came primarily from concerns about retention in the major (on average 32% of majors were leaving the major within a year). We selected a media computation approach based on a literature review of the research and experiences of others using non-traditional approaches. We sought an improved approach that both supported a comparable level of academic rigor and "similar enough" concepts/content coverage in the Java programming language (enabling students to go directly into the next course).

Courses ranged in size from 47-176 with an average of 83 for the traditional course and 131.5 for the media computation course. In general, the structure and workload of the class was kept the same: weekly 50 minute supporting closed labs, 8-9 programming assignments (done using pair programming), weekly quizzes, one midterm, one final. Another gauge of the workload of the course comes from students' self-reported number of hours spent studying outside of class: the traditional course had an average of 8.8 hours/week but the media computation course was lower with 6.4 hours/week. One notable difference between the courses was that the media computation course was taught using Peer Instruction in lectures, and the traditional course was not (though it often engaged students in in-class mini-activities in the same "style" as Peer Instruction). It is hard to assess how this difference affected student learning. However, Peer Instruction has been shown to increase learning in physics courses [1].

# 4. RESULTS

We report two key issues, which inform about the differences in the traditional verses media computation approach: desired competencies (measured via programming assignments and exam questions) and student success (measured via course pass rate and retention in the major one year later).

In the analysis of desired competencies we analyze data from one term each year (Winter) for clarity. We report on the content and knowledge required in the assigned work, e.g. in programming assignments and final exams. Anecdotally, the differences between terms in each year were minimal, but the programming assignments varied in the traditional quarters (though they were identical in the media computation quarters).

**Table 1. Course Grade Comparison**

| | **Traditional** | **Media Computation** |
|---|---|---|
| Winter Term Course Grades | $\bar{X}$ =79.9%<br>s.d. = 12.1 | $\bar{X}$ = 83.4%<br>s.d. = 14.0 |

**Table 2. Comparison of concepts/constructs in programming assignments. T is traditional approach, light gray is media approach.**

| PA\Week | 1 | 2 | 3 | 4 | MT | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| If statements | | T | T | | | T | T | T | T |
| Loops | | T | T | T | | T | | T | T |
| 1-D Arrays | | | | | | T | | | |
| Nested loops | | | | T | | | | | |
| 2-D Arrays | | | | | | | | | T |
| Object Use | | T | T | | | | T | | T |
| Class Design | | | | | | | T | T | T |

We do not focus our analysis on course grades, because, in practice, there are many factors which influence course grades. Although the average grade in the media computation course (Table 1) in Winter 2009 was statistically significantly (two-group t-test, p=0.0203) higher than the traditional course in Winter 2007, this difference (3.5%) is not very meaningful in practice. However, the similarity of course grades implies similar student performance on the programming assignments and exam questions analyzed in the following section.

## 4.1 Desired Competencies: Programming Assignments

Although one could use a detailed analysis of textbook content and ordering to compare a traditional CS1 with a media computation approach, it is easy to become overwhelmed by details and minor constructs. We choose to instead analyze programming assignments (assigned weekly) because of the specific emphasis the instructor took each week to have the programming assignment engage the students deeply with the key concepts and constructs. By looking at the ordering and coverage of key introductory concepts in Table 2, experienced instructors can get a high level view of some of the key differences one can expect in teaching a media computation based CS1.

The kinds of assignments one develops for a media computation course are notably different than in a more traditional course. The instructor attempted to make traditional assignments as interesting and relevant as possible, e.g. asking students to add key movement methods in a Tetris game, Caesar cipher, managing contacts in a cell phone, and simulating Amoeba population growth (given a graphical interface). However, creating such assignments (and changing them a bit each term) was highly time-consuming. Additionally, some standard, less interesting assignments persisted including calculating vending machine change, finding min/max/average, managing class grades, and implementing an Odometer class.

In contrast, developing media computation assignments was relatively easy (could be quickly "imagined" as variations, augmentations, or similar Picture or Sound modifications to what

**Table 3. Media Computation Programming Assignments**

| PA | Assignment |
|---|---|
| 1 | Draw your first name (with Turtle object) |
| 2 | Draw 5 nested shapes (Turtle)<br>Create a Picture with every other pixel green or black |
| 3 | Modify a picture to reduce some percent of each color component (based on parameters)<br>Modify the top, middle, and bottom third of a Picture based on three different filters (two copied in from book, one of your own design). |
| 4 | Create a simple collage with three copies of an image, all with at least one filter applied (one of your own design)<br>Modify a picture to flip horizontally and vertically a portion of the picture |
| MT | Create a collage using at least three Pictures with each having at least one filter applied (a smaller assignment during midterm week but that let students be less constrained than the previous collage) |
| 5 | 3-way chromakey (replace background, replace shirt color) |
| 6 | Re-do picture flip (4) but error check for parameters that cause out of bounds errors<br>Make a new Sound by concatenating two Sounds |
| 7 | Make a song (a collage of Sounds) using at least 4 methods (reverse, changePitch, one of a set we provided, and one of the student's own design) |
| 8 | Design a class (not media computation based) |

was in the book), and many had highly creative and individualized results which allowed them to be used in multiple terms with little fear of copying from one term to the next. The weeklong assignments used in Winter are shown in Table 3 (some assignments had two smaller problems). We intentionally created challenging assignments that focused students on novel problem solving and could not be solved by copying (or making a minor modification to) code in the book. In hindsight, traditional assignments seemed, by comparison, contrived and much less focused on challenging problem solving than they were on getting students to use a specific Java construct/concept or feature.

## 4.2 Desired Competencies: Exam Questions

In addition to student practice with concepts and constructs in programming assignments, in the end, the final exam plays a major role in determining whether the student has passed the course, and "is ready" to take the subsequent course.

### *4.2.1 Comparison in Concept/Construct Coverage*

Although it would be interesting to repeat specific exam questions from the traditional courses' exams in a media approach, it becomes practically challenging (due to the specifics of each approach). However, in Table 4 we compare two final exams (Winter term of each year) for coverage (in terms of % of points) of basic concepts/constructs. In doing so, we noted some questions (on the traditional exam) covered no concepts/constructs (7 questions accounting for 24% of points). These questions often tested knowledge of language issues (semantics, rules). We additionally report the percentage of

**Table 4. Comparison of concepts/construct in examination points/marks.**

| Concept/Construct | Traditional | Media Computation |
|---|---|---|
| If statements | 39% | 41% |
| Loops | 32% | 35% |
| 1-D Arrays | 31% | 43% |
| Nested Loops | 10% | 56% |
| 2-D Arrays | 6% | 40% |
| Object use | 4% | 74% |
| Class Design | 37% | 7% |
| Language Issues | 47% | 6% |

points on each exam where language issues were being tested (sometimes in conjunction with concepts/constructs). Note, percentages may add up to over 100% due to questions covering more than one area (reported below).

It is notable that the basic constructs/concepts categorized here occur more frequently in the media computation exam, with the exception of class design. This may be partially due to the fact that the traditional exam had much more emphasis on language issues (discussed further in 4.2.2).

In general, a question that requires use of more than one construct might be considered more challenging that one which requires only understanding and application of one. For example, a loop problem with an array may be more complex than a simple loop. The addition of an if-statement to that question may make it even more complex. In analyzing exam complexity based on required combinations of construct/concept use, we chose to not consider object use since it seemed to be a small part of the difficulty of the question. Similarly, we ignore class design because it appears primarily orthogonally to the rest of the concepts/constructs analyzed. Using this approach we find that 83% of the points on the media computation exam require use of two or more concepts, compared to 37% of points on the traditional exam. Arguably, having to employ more than one concept to solve a question does make that question more challenging; so, the media computation exam is notably more challenging. However, as mentioned in introduction to this section, there was no statistically significant difference in grade distributions between the Media computation and traditional students.

### *4.2.2 Exam Question Styles*

The final exam structure was very similar both years in terms of styles of questions asked. These included code tracing questions, select a line of code to complete a method, select a code fragment to complete a method, select appropriate method header or parameter list (for standard class design), explain what a code fragment does in English [5], and write code. In the traditional year, there were also a number of language issue questions. Such questions asked about issues such as legal overloading, the difference in comparing Strings with == versus .equals, details of constructor design (including issues of aliasing with arrays), and differences in parameter passing (call-by-value) as applied to primitive and class type variables.

However, while question styles may have been similar, surface features of media computation exam problems differed a great deal from traditional exam problems. Media computation exam questions, for example, asked students to select a picture that

would be generated (or to describe the picture generated) when a method is called on a specific input picture. The questions described a desired Sound (or Picture) modification to be implemented in a method and asked students to write or select correct lines of code to fill in template code. In Winter 2009 specific questions included blurring a Picture (by averaging the surrounding Pixel values), morphing a grayscale Picture to black and white, and putting a variable width "frame" around the outer edge of a Picture. Students were asked to write code on the exam on the following topics (not described completely):

- Write a method of the Sound class that creates a Sound by taking a Sound and repeating it a specified number of times (passed as a parameter).
- Write a method of the Picture class that modifies a Picture by mirroring the right third of the image into the left third – however, only every other column should be mirrored.

#### *4.2.3 Ability of Media Computation Students on non-media context questions*

In Winter 2009, the instructor became particularly interested in whether students could perform comparable tasks on non-media context problems as they had (or were being tested on) with media context. As a result, 13 questions accounting for 25% of the points on the exam asked questions that could appear on a non-media computation course final exam – that is they use basic types (int, double, Strings) and arrays of those types. However, these questions were only included in multiple-choice questions (code tracing, code selection, code completion, code analysis) where students would hopefully be less impacted by syntax (as they might be in code writing). As shown in Table 5, students, in general, did quite well on these questions, especially given their extremely limited exposure to non-media context Java code.

**Table 5. Percent of students correctly answering non-media context exam questions (average by topic).**

| Question | Correct |
|---|---|
| If statement design (2Qs) (when to use if, else-if, else, compound if statements) | 94% |
| Nested Loop iterator (2Qs) (trace) | 92% |
| Class Design/language issue (5Qs) (select code) | 85% |
| Find index of max element in array (1Q) (select code fragment) | 74% |
| Loop over array replacing every other element with its index (1Q) (trace) | 69% |
| Complex array reverse, with two iterators (1Q) (trace) | 67% |

### 4.3 Student Success: Pass Rates and Retention

Our second key interest was in the impact of the media computation approach on majors' success in our CS1 course. We measure this in two ways: the pass rate of the CS1 course itself and the retention of majors in the major one year later (measured as whether students were still enrolled in a CS course one year later). Additionally, we report the general pass rate of the entire class (both majors and non-majors) for comparison with previous studies of non-majors.

**Table 6. Student Success Measures**

| | Traditional | Media Computation |
|---|---|---|
| Major Course Pass Rates | 86.8% (99/114) | 92.9% (159/171) |
| Course Pass Rates (majors and non-majors) | 80.4% (131/163) | 90.1% (237/263) |
| Retention Rate of Majors 1 year later | 62% (67/108) | 71.1% (27/38)** |

**Only reported on Winter 2009 term, Fall 2009 term data is not yet available.

We preformed a two proportion Z-test to compare the media computation and traditional data in the above table. This revealed that both the overall pass rate ($p=0.0021$) and the pass rate for just majors ($p=0.0414$) showed statistically significant improvement in the media computation courses. The improvement in retention of majors one year later did not prove to be statistically significant ($p=.1591$). This may be due to differences in student demographics between the Winter and Fall terms; we only had Media Computation retention data for Winter 2009. However, if we compare only the two Winter terms we do see statistically significant improvement in retention ($p=0.0476$).

## 5. DISCUSSION

Based on the above analysis, it seems clear that students in a media computation-based CS1 designed for majors do cover (and learn) comparable materials to students in a traditional approach. Although the order in which concepts/constructs are introduced is notably different, analysis of exam question content shows the media computation approach giving greater emphasis to all core concepts, except for class design. Additionally, we find media computation exam problems to be more complex than traditional problems, when we define complexity as requiring the application of more than one concept on a question.

But, in fact, something does "give" to allow the media computation approach to focus so much more on the application of core concepts to problem solving. It is clear from exam analysis that student experience with language issues and features is much reduced. Some of the specific semantics and features missing (or notably less emphasized) include overloading, 2-D array indexing syntax (students use 2-D arrays via an accessor method), and parameter passing (specifically differences between passing primitive and reference parameters). Additionally, in our 10-week course students got no experience with textual File I/O (e.g. Scanner), nor with String methods, relatively less experience building their own classes, including specifically the complexities of managing array instance variables. Somewhat surprisingly (with the greater emphasis on loops and arrays), students get little or no experience with some standard data analysis patterns such as find minimum or average. They also had less experience with testing (e.g. different types of inputs) and designing test cases since programs are often designed to work with a particular Picture or Sound file. It should be noted that some of these issues are, in fact, present in the textbook. But the instructor found these often seemed a distraction from the overall problem solving focus, and quickly dropped them (except in passing). A second note is that an interesting chapter in the media computation textbook has

been adopted by the second 10-week course (CS1.5) and covers File I/O, String methods, overloading (via String methods), and ArrayLists.

So what filled in the time not spent on the above issues? The overriding experience of the instructor was that most course time was focused on problem solving. Each week involved the proposal of a new problem – how would you get this effect in a Picture or in a Sound? Then an appropriate concept or construct is introduced to solve that problem. Students spent more time being presented with problems, which required analysis and thoughtful development of code (in comparison to rote implementation to practice language features). This led to increased student experience specifically with indexing into arrays, performing complex array element manipulation, looping (of all kinds), and working with complex objects (digital representations of media). Additionally, some common programming patterns are present – including swap (via reversing a Picture or Sound). One important feature that is not revealed in the previous analysis is that once students learn concepts/constructs with Picture objects (~7 weeks), they then re-visit them in the new context of Sound objects (~2 weeks) – having the chance to reinforce and expand their understanding of those concepts/constructs. But it is the, perhaps hard to define, increased student engagement in complex problem solving that seems most notable at the end of the day.

## 5.1 Instructor Experience

Teaching a media computation CS1 course was definitely a positive experience for the instructor. One of the greatest joys was to remove the feeling of dragging students through one Java feature to another, including sometimes arcane issues that were too often described to them as "you'll really find out why you need to do this later". This was replaced with presenting and solving with students interesting problems that one could share with others (we hold an art show and post student work on a wiki.) The content of the course itself is exciting, perhaps ameliorating the importance of having a CS1 instructor "who is really engaging" (and can make finding vending machine change sound cool).

An incredible benefit to the instructor is that programming assignments do not need to be reinvented every term to deter plagiarism among students (because the solutions are creative – there's not just one). Additionally, creation of assignments was easier, following an extension or variant of something in the book.

There are important issues that can cause trouble for an instructor beginning to teach with media computation. Critically, students experience language features in a dramatically different order than "usual". For example, this instructor accidentally required creating and returning a Picture object long before it was introduced in the book. Additionally, if statements aren't covered until week 5 or 6. It's amazing what you can do without them. Finally instructors should be prepared to create entirely new exams, as questions will now ask about Pictures and Sounds. These questions can be challenging to describe in English and often need to provide a sample input and output image to adequately describe the intended problem.

## 5.2 Student Experience

As part of a larger study, we have interviewed 39 students about their experiences in a media computation CS1. While those results are beyond the scope of this paper, a few positive preliminary experiences stand out. Students mention being proud of their work, saying "look at how cool they [pictures] looked… I would show my parents and show my friends."

Students also positively note the connection of homeworks with the real world (e.g., Photoshop):

> "it was something that you'd seen before and you're just like how does that work and then you figure out wow, that's how it works. It was actually pretty cool just to figure out…This is how something in the real world that we see every day really works and this is how it's done."

Others enjoyed real value from their assignments -- one student told us his family wanted to use his Sound collage as a ringtone.

# 6. CONCLUSIONS

We report on our experiences implementing a CS1 course for majors with a media computation context. We compare required student competencies in the new course to our traditional approach via analysis of programming assignments and exam questions. While concepts and constructs are introduced in a non-traditional order, they feature prominently in programming assignments. Exam question analysis reveals increased emphasis on core concepts of loops, arrays, if statements, and object use in the media computation approach and less emphasis on class design and language issues. Course pass rates (both for majors and the class as a whole) are significantly higher in the media computation course, though strong evidence of increased retention of majors (one year later) cannot yet be reported.

# 7. ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under NSF CNS-0933635.

# 8. REFERENCES

[1] Crouch C. and Mazur, E. Peer Instruction: Ten years of experience. *Am. J. Phys. 69 (9) 2001.*

[2] Forte, A., Guzdial, M. 2004. Computers for Communication, Not Calculation: Media as a Motivation and Context for Learning. Proc. of Hawai'i International Conf. on System Sciences, January, 2004.

[3] Forte, A., Guzdial, M. Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses. IEEE Trans on Education, 48(2) 2005.

[4] Guzdial, M., Ericson, B. *Introduction to Computing and Programming with Java: A Multimedia Approach.* Prentice Hall, 2007.

[5] Lister, R, Simon, B., Thompson, E., Whalley, J., Prasad, C. Not Seeing Forest for the Trees: Novice Programmers and the SOLO Taxonomy. ITiCSE 2008.

[6] Margolis, J., and Fisher, A. *Unlocking the Clubhouse: Women in Computing*. MIT Press, 2003.

[7] Rich, L, Perry, H., Guzdial, M. 2004. A CS1 Course Designed to Address Interest of Women. SIGCSE 2004.

[8] Sloan, R., Troy, P. 2008. CS 0.5: A Better Approach to Introductory Computer Science for Majors. SIGCSE 2008.

[9] Tew, A., Fowler, C., Guzdial, M. 2005. Tracking an Innovation in Introductory CS Education from a Research University to a Two-Year College. SIGCSE 2005.