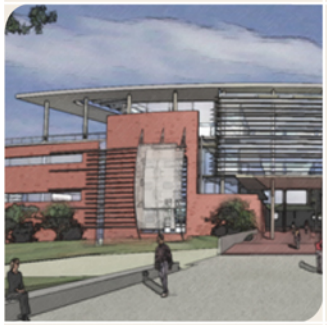
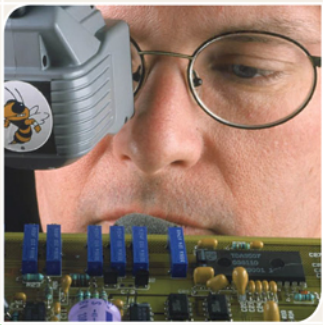


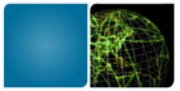



Technology for Teaching the Majority about Computer Science

Mark Guzdial
School of Interactive Computing

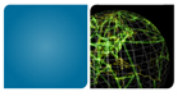
Georgia Tech College of Computing

Story

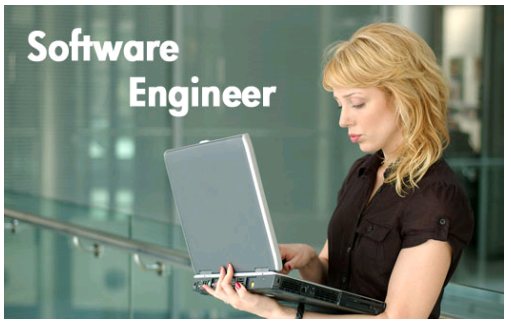



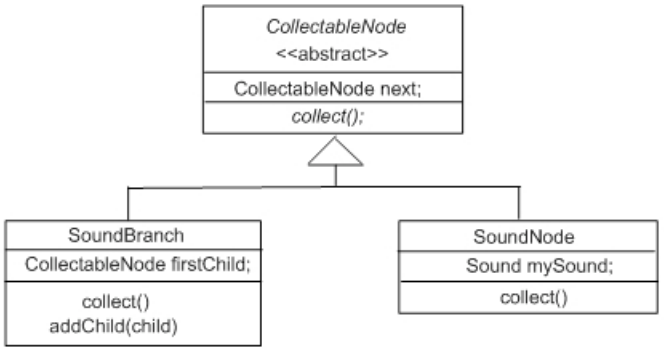

- Computing is important for more than just those who choose to major in computing.
  - Who are those (the majority) who need computing but won't major in computing? What do they need from CS, and why aren't they in our classes? How do they now learn CS?
- Teaching those who do *not* want to become software engineers or computer scientists
  - The Story of Computing for All at Georgia Tech.
- Meeting the needs of the majority, with technology:
  1. Matching or tailoring the context to the person.
  2. How to find what they *really* want and need.
  3. Teaching CS concepts without apprenticeship.

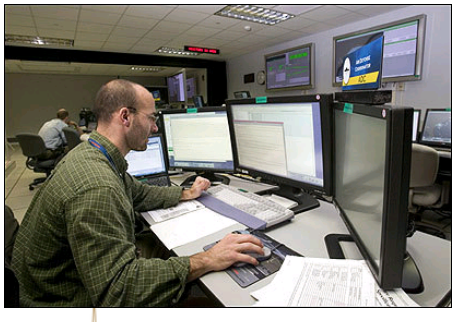
The typical CS student: Future Software Engineer

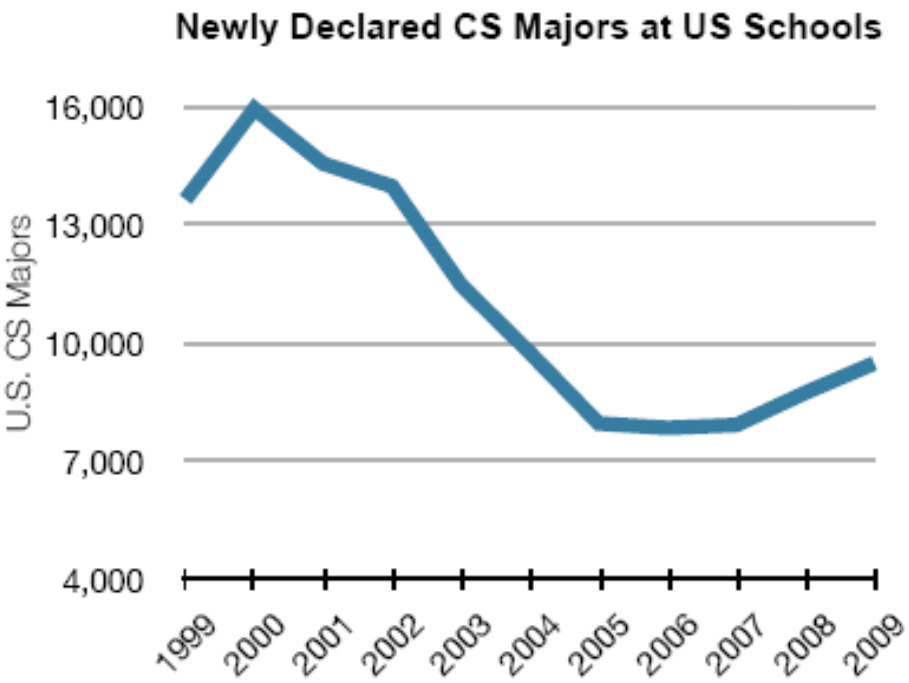

- To produce reliable, robust, secure software.
- To work in interdisciplinary teams.
- To use appropriate design notations, such as UML.
- To work in multiple programming languages.



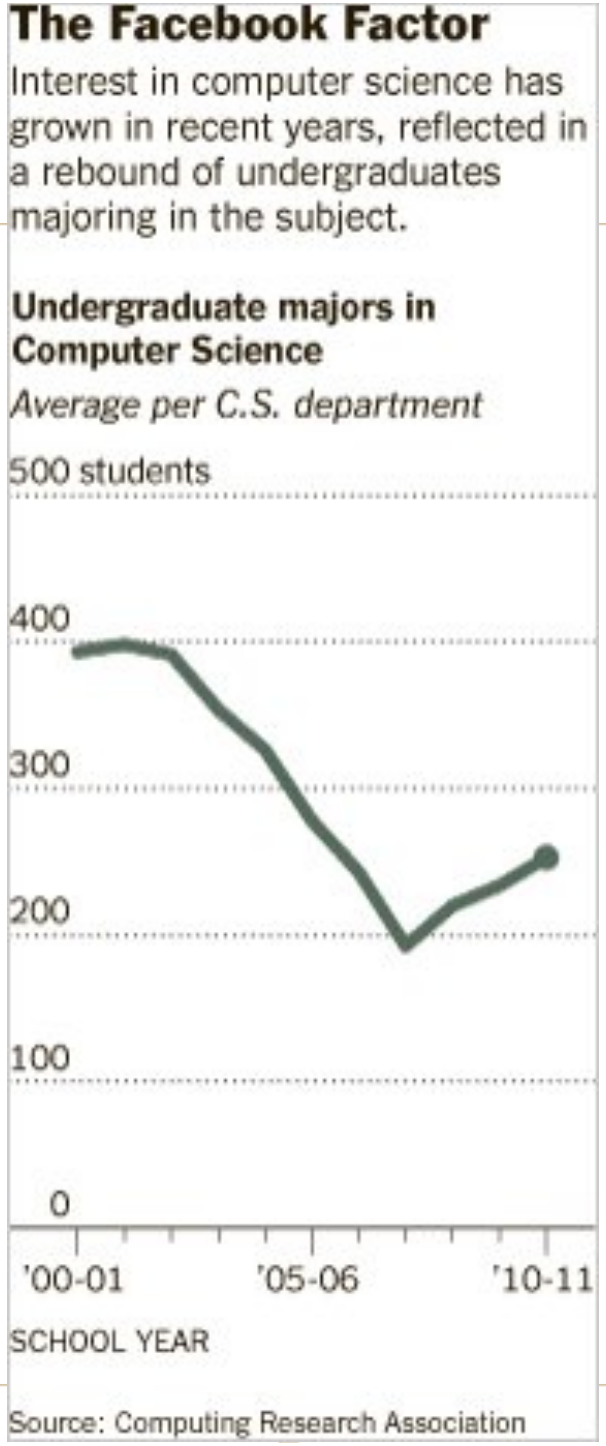




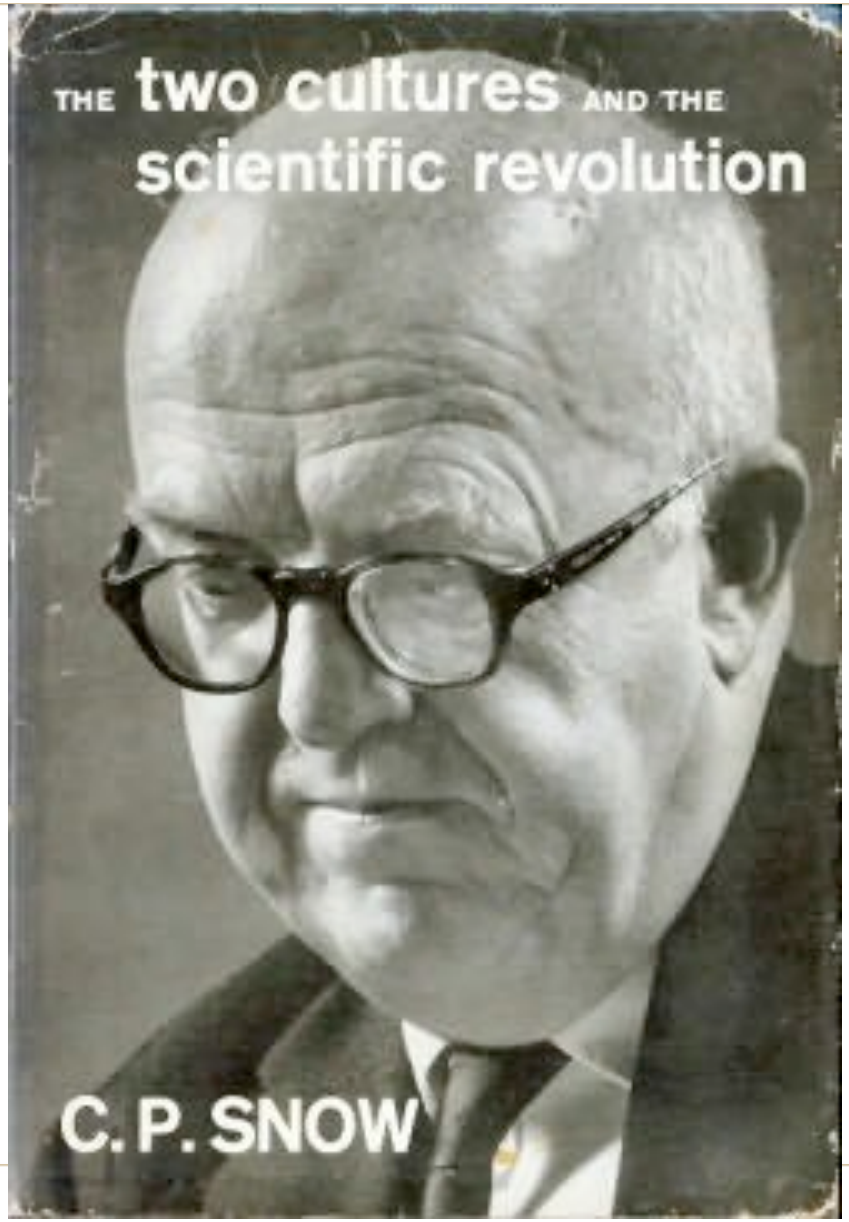
Taulbee Numbers

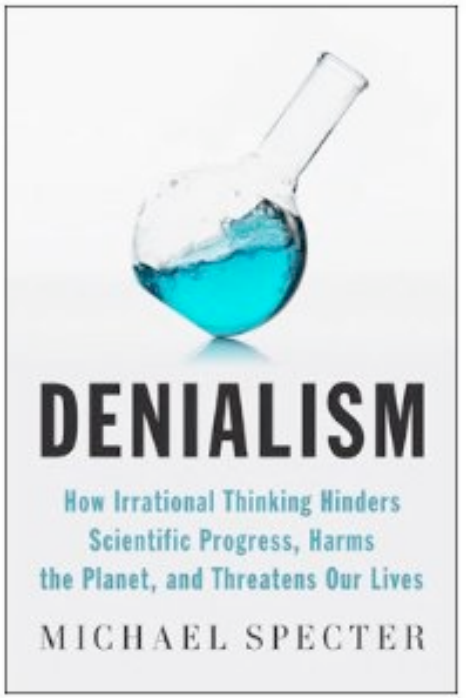


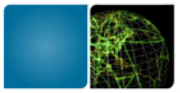
2010 CRA Taulbee Survey of PhD-granting institutions



The Two Cultures





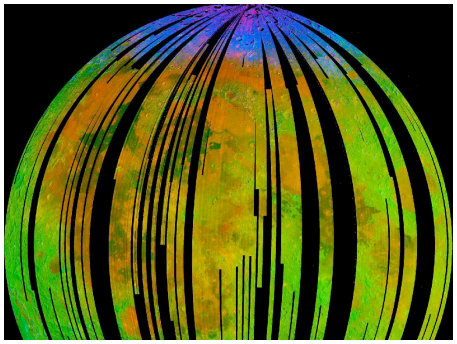



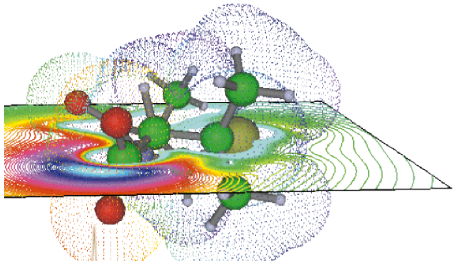

Who else wants what CS has to offer?

- Computing is at the core of the modern society and modern economy.
- Computing is key to innovation in many disciplines.
- ***Computer Science has a much larger potential audience beyond software developers.***
  - Estimates:
    ~13 million non-professional programmer/end-user programmers in US by 2012,
    vs.
    ~3 million professional software developers (Scaffidi, Shaw, & Myers, 2005)

An atypical CS student: Future computational scientist or engineer

- To use computation as a tool to enhance understanding.
- To write programs of (at most) 100 lines (most often, 10 lines) for themselves.
  - They care about the products of the programs, not the programs.
- To learn as few languages as are needed for their tasks.
- To work in interdisciplinary teams, including software engineers.



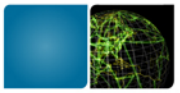


An atypical CS student: Future high school CS teacher

- To use code to explore and understand ideas of computation.
- To learn what languages are necessary to meet standards and engage students.
- To work with students with a wide range of interests.
  - Probably won't work with professional software engineers

Georgia Tech

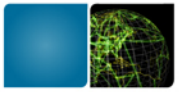
An atypical CS student: Future graphics designer


- To write programs to improve their efficiency, and to implement their dynamic (e.g., Web) designs.
- To do as little coding as possible.
- To learn about computing ideas in order to improve their process, but with a focus on people and creativity.
  - Probably won't work with professional software engineers



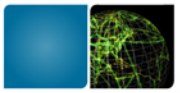




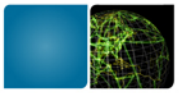

How do meet this need?

- Our track record for the first CS course is poor.
  - 30-50% failure or withdrawal rates (Bennedsen & Caspersen, 2007)
- Other majors tend to be more female and more ethnically diverse than the typical computing student.
  - Our track record with these audiences is particularly poor (Margolis & Fisher, 2003)

Technology can help. We should lead.

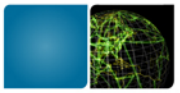

- As computer scientists (and CS teachers), we can and should *lead* in developing using technology to improve learning.
  - Unlike most teachers, we can build our own technology.
  - Done well, our solutions may inform other disciplines.
- We need technology to support CS learning for the majority.


Three Needs

- First, the importance of context in teaching computing.
  - Need #1: Match or tailor context to make sense to students.
- Second, why aren't the majority looking to our classes in CS? Where *are* they looking?
  - Need #2: Help "amateur" (non-CS professional) programmers find the help they need.
- Third, teaching CS with less programming.
  - Need #3: Create new kinds of instructional materials for a new pedagogy of computing education.

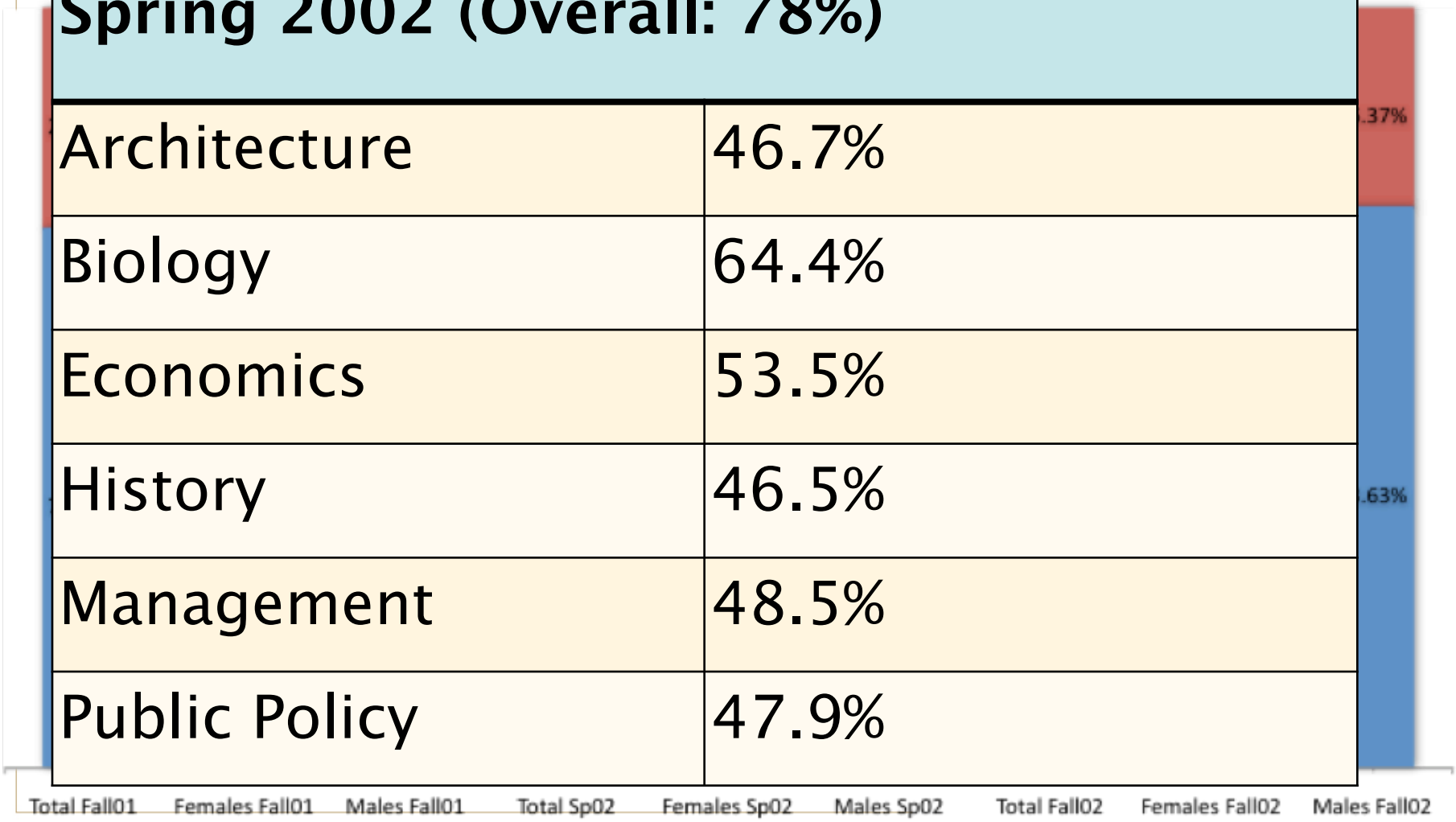
Piece 1: Teaching Computing to Everyone


- *Fall 1999*:
  All students at Georgia Tech must take a course in computer science.
  - Considered part of General Education, like mathematics, social science, humanities…
- 1999-2003: Only one course met the requirement.
  - Shackelford's pseudocode approach in 1999
  - Later Scheme: How to Design Programs (MIT Press)

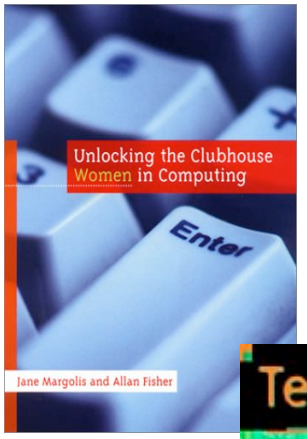
One-class CS1: Pass (A, B, or C) vs. WDF (Withdrawal, D or F)

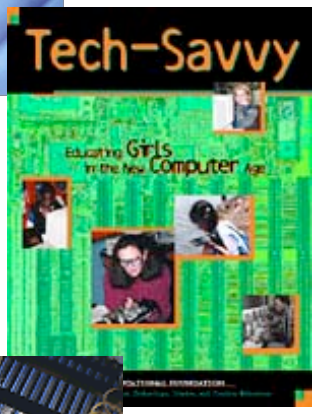
Success Rates in CS1 from Fall 1999 to Spring 2002 (Overall: 78%)	
Architecture	46.7%
Biology	64.4%
Economics	53.5%
History	46.5%
Management	48.5%
Public Policy	47.9%

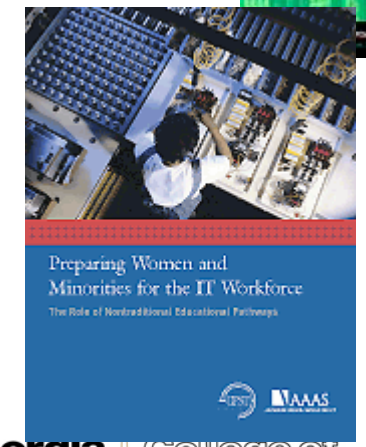


Contextualized Computing Education

- What's going on?
  - Research results: Computing is "tedious, boring, irrelevant"
- Since Spring 2003, Georgia Tech teaches three introductory CS courses.
  - Based on Margolis and Fisher's "alternative paths"
- Each course introduces computing using a context (examples, homework assignments, lecture discussion) relevant to majors.
  - Make computing relevant by teaching it in terms of what computers are good for (from the students' perspective)



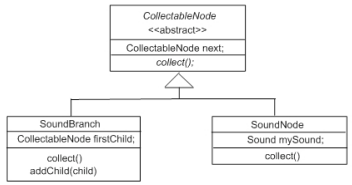


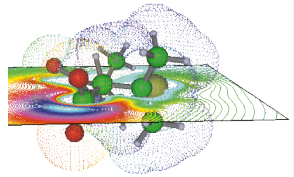


Georgia Tech College of Computing

Our Three CS1's Today

- CS1301/1321 *Introduction to Computing* Traditional CS1 for our CS majors and Science majors (math, physics, psychology, etc.). Now, uses robots.
- CS1371 *Computing for Engineers* CS1 for Engineers. Same topics as CS1301, but using MATLAB with Engineering problems.
- CS1315 *Introduction to Media Computation* for Architecture, Management, and Liberal Arts students.

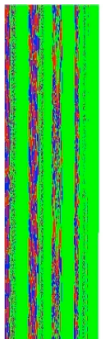




Media Computation: Teaching in a Relevant Context

- Presenting CS topics with media projects and examples
  - Iteration as creating negative and grayscale images
  - Indexing in a range as removing redeye
  - Algorithms for blending both images and sounds
  - Linked lists as song fragments woven to make music
  - Information encodings as sound visualizations



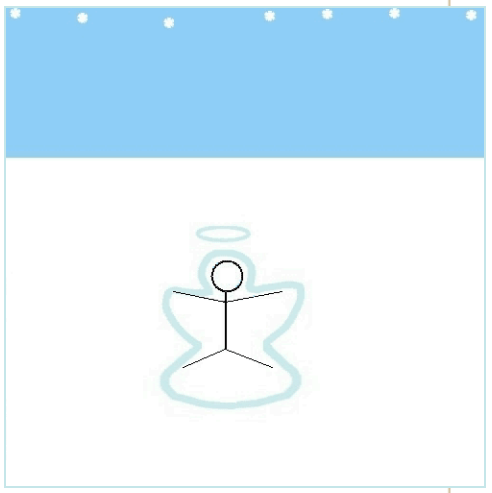



```
def clearRed(picture):
  for pixel in getPixels(picture):
    setRed(pixel,0)
```

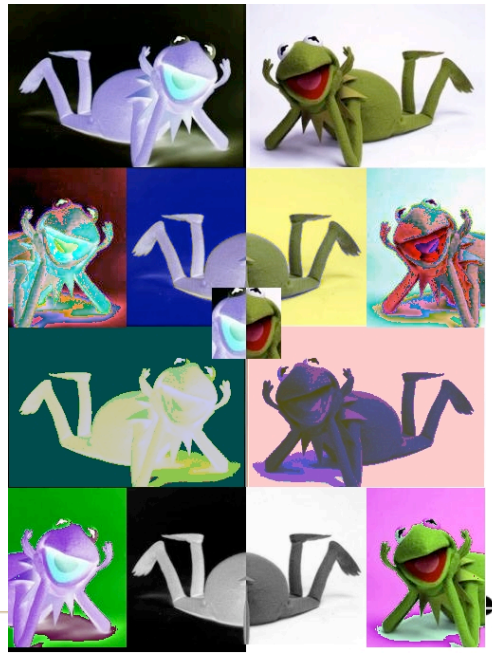
```
def greyscale(picture):
  for p in getPixels(picture):
    redness=getRed(p)
    greenness=getGreen(p)
    blueness=getBlue(p)
    luminance=(redness+blueness+greenness)/3
    setColor(p, makeColor(luminance,luminance,luminance))
```

```
def negative(picture):
  for px in getPixels(picture):
    red=getRed(px)
    green=getGreen(px)
    blue=getBlue(px)
    negColor=makeColor(255-red,255-green,255-blue)
    setColor(px,negColor)
```

Open-ended, contextualized homework in Media Computation CS1 & CS2



Sound collage

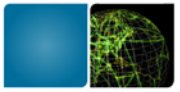


Results:CS1“Media Computation”

Change in Success rates in CS1 “Media Computation” from Spring 2003 to Fall 2005 (Overall 85%)		
Architecture	*46.7%*	85.7%
Biology	*64.4%*	90.4%
Economics	*54.5%*	92.0%
History	*46.5%*	67.6%
Management	*48.5%*	87.8%
Public Policy	*47.9%*	85.4%

WDF
Pass

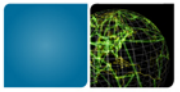
Was the class successful?


- Interviewed Intl Affairs student (female): “I just wish I had more time to play around with that and make neat effects. But JES [IDE for class] will be on my computer forever, so… that’s the nice thing about this class is that you could go as deep into the homework as you wanted. So, I’d turn it in and then me and my roommate would do more after to see what we could do with it.”

- **Survey one year later:**
  - 19% of respondents had **programmed** since class ended

- **"Did the class change how you interact with computers?”**
- “Definitely makes me think of what is going on behind the scenes of such programs like Photoshop and Illustrator.”


Results at Other Schools

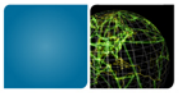
- Similar retention results at 2 year public Gainesville College (Tew, Fowler, Guzdial, SIGCSE 2005) and at (much more diverse) U. Illinois-Chicago's CS0.5 (Sloan & Troy, SIGCSE 2008)
- Would you like more CS?
  - GT 15.2% "Strongly Disagree." <25% agree.
  - More MediaComp? GT and Gainesville over 40% agree.

Table 2. Success rates at Gainesville College before and with Media Computation class.

	ENROLLMENT	SUCCESS RATE
Gainesville's CSCI 1100		
Average 2000 – 2003	28	70.2%
Media Computation		
Summer 2003	9	77.8%
Fall 2003	39	84.6%
Spring 2004	22	77.3%
Summer 2004	11	90.9%

Our school's CS 0.5	Enrollment	Success Rate
Fall 2002	61	74.8%
Spring 2003	38	76.7%
Fall 2003	51	68.6%
Spring 2004	22	82.9 %
Fall 2004	15	93.3 %
Average "Old"	37	**75.9%**
New Spring 2005	18	94.4%
New Fall 2005	29	90.0%
New Fall 2006	42	76.2%
New Spring 2007	24	83.3%
Average "New"	28.3	**84.1%**

Table 1: Success rate with CS 0.5 before and with new approach. Averages weighted by enrollment.


Results at University of California, San Diego

- Using Java Media Computation as normal CS1 for CS majors at a research university.
- Did extensive data collection last semester before switching to Media Computation.
- Been following two cohorts of CS1 students for comparison.

Simon, Kinnunen, Porter, Zaskis, ACM ITICSE 2010

Findings:

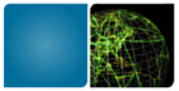
• MediaComp has more focus on problem-solving, less on language.

• MediaComp students have higher pass rates and retention rates one year later

A Contextualized CS2: MediaComp Data Structures

How did the Wildebeests charge over the ridge in Disney's "The Lion King"?

Research Question:
Is context still useful in a second course?

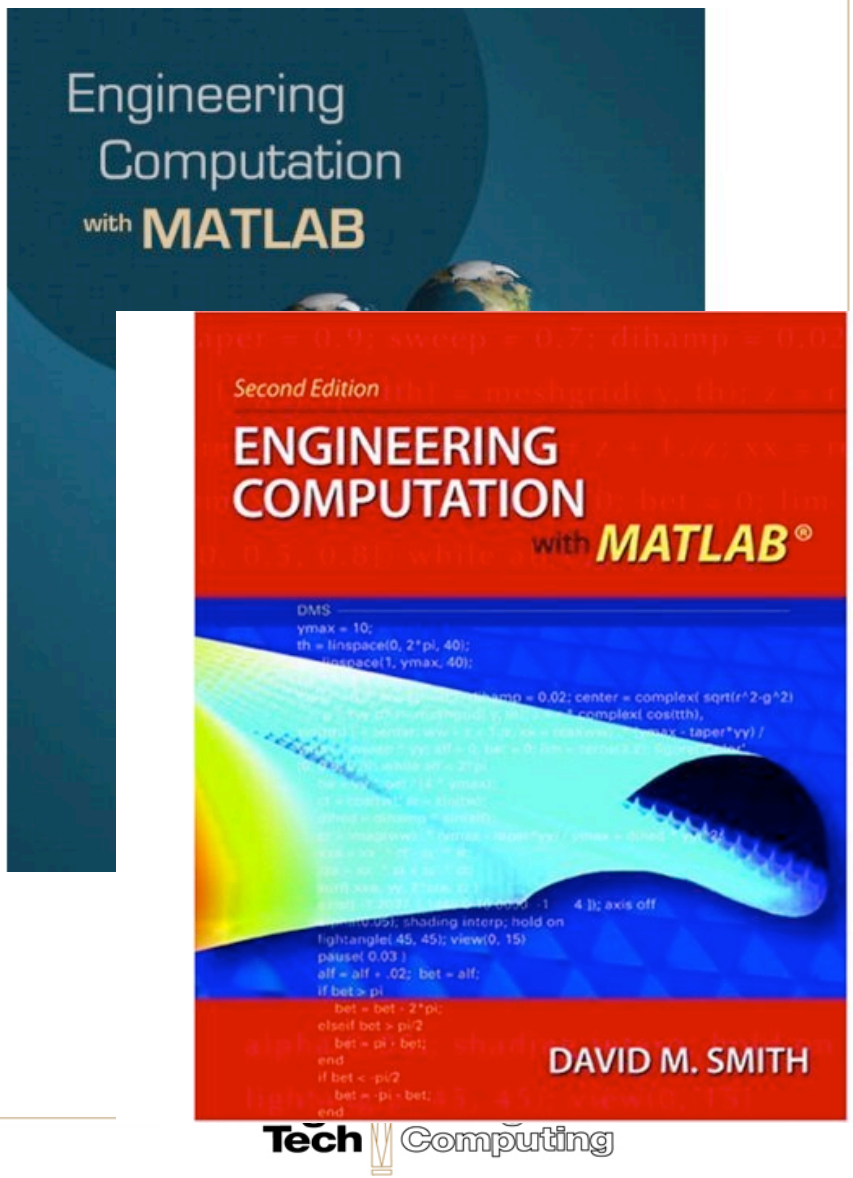

- Mixed model research design
  - Interviews informing whole-class survey
- 11% agreed with "Working with media is a waste of time that could be used to learn the material in greater depth."
  - "I didn't take this class to learn how to make pretty pictures."
- A majority of the class (70%) agreed or strongly agreed that working with media makes the class more interesting.
- 67% of the students agreed or strongly agreed that they were really excited by at least one class project
- 66% reported doing extra work on projects to make the outcome look "cool."

(Yarosh and Guzdial, JERIC, Jan 2008)

Introducing Computing in an Engineering Context

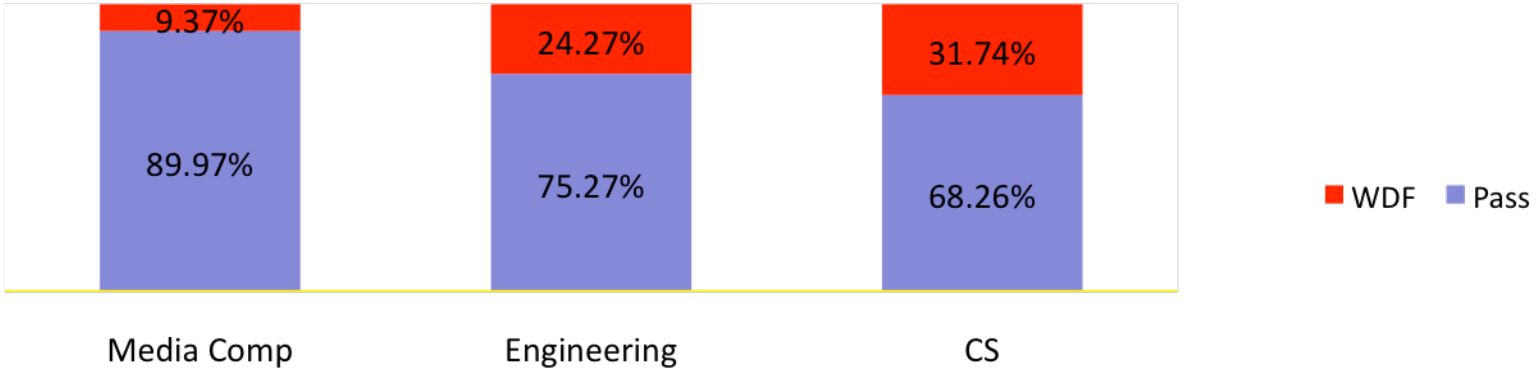
- Developed in collaboration with Civil, Mechanical, and Aerospace Engineering.
- Uses Engineering problems and MATLAB
- Covers traditional CS1 topics

- Among our 3 CS1's, these are the first students to program outside of class.
- The success rate in this class also rose compared to all-in-one.



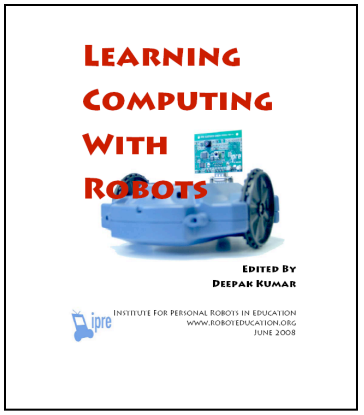
Comparing Spring 2004

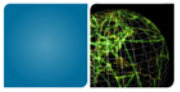
CS for Engineers: ~1200 students/semester
Media Comp: ~300 students/semester
CS for CS majors: ~150 students/semester



A Context for CS1 for CS majors: Robotics

- Microsoft Research has funded the Institute for Personal Robotics in Education
  - Leads: Tucker Balch, Deepak Kumar, Doug Blank
  - Joint between Bryn Mawr College and Georgia Tech
  - http://www.roboteducation.org
- Developing a CS1 with robotics as the context.

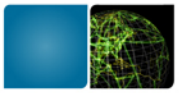



Need #1: Finding or Making Context

- Context *is* important.
  - It works to support success and engage students.
  - But, there is no universally successful context.
    - At Georgia Tech, there are continued demands for finer granularity (e.g., business, biology).

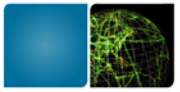
CS Teaching Need #1:
Help students and teachers to match interests and abilities to contextualized educational material, *or*
From a general form, instantiate template into a given context (e.g., sampling, sorting, searching)


CS Need #1: Matching or Tailoring

- **Matching** students to existing contextualized content.
  - What variables are important? Level, major, desired degree, previous background.
  - More than a search engine.
- **Tailoring** content from descriptions of context-specific features to learning objectives.
  - Pixels in pictures and samples in sounds: Sampling
  - Discrete event simulation queues: Sorting

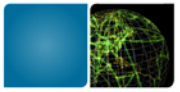
Piece 2: What do they need that we have to offer?


- Brian Dorn studied graphics designers who program.
- Conducted a series of interviews and assessment activities.
- Found that these subjects want more computer science, but don't find courses (and most other resources) adequate (Dorn & Guzdial, ICER 2010)

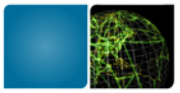
- *P10: So, that was a really long way of saying* ***yes, I think that an academic study would make me a better programmer, but not by a whole lot.***

What do software engineers do? Answer: The Boring Stuff.


- P2: I was able to take different samples from different places and instead of just being let's say an MIS major, or computer science major, you know it's—**you're not going to be front-end anything with computer science. You're going to be back-end everything.**

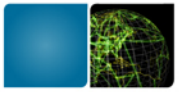
- P4: I think as **a front-end developer**, you focus more on the design and the usability, and you're focusing more on the audience. And then on the **back-end** I think you're focused on more, these are like the software developers. **And they're programming something, and they don't really see what it's gonna look like; they're just making it work.**


Why don't they take CS classes?

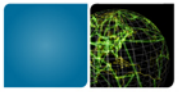
- P7: **I started out in computer science, but didn't like it at all.** The fact that I wasn't learning anything new. I took an intro to programming course, and then I talked to some other people in the program and it was all repetition and I guess there wasn't any really new**. So you weren't really learning any concepts. You were learning the languages, and I didn't like that at all. So that's why I left…**

- *Do* we just teach *languages*?
  Why don't they *see* the *concepts*?


They are not afraid of coding

- “What interests you about web design?”
- P12: The coding! I don't like to code. But the things that the code can do is amazing, like you can come up with this and voila, you know, it's there. Javascript for one. The plugins and stuff. I think that's very interesting, intriguing and stuff. Because I mean like the code is just, there's so much you can do with code and stuff. It's just like wow.


They're Lost without Initial Knowledge

- They learn from websites, reading lots of code, books where they can, friends.
  - Rarely courses.
- Surprising tidbit: Learning less than they might because of a lack of deep knowledge.
  - For example: Exploring code by searching Google for function and variable names.
- **Brian's experiment:** Given a case library **with** conceptual information vs. a code repository **alone,** what gets learned, used, and liked? (ICER 2011)

Script Development

The goal of this project is to automatically disable all of the text layers in a mockup. We'll begin by thinking about how we would accomplish this manually in the Photoshop interface. The process is pretty simple.

1. Consult each layer in the layer palatte one at a time.
2. If a given layer is a text layer (i.e., the layer's icon shows the letter "T"), disable it by clicking on the eyeball icon at the left.

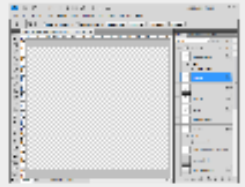
To translate this into a program, we need to first find out how to access the layers in our program. Using the Object-Model-Viewer, we can determine that the layers of the current document in Photoshop can be accessed using `app.activeDocument.layers`. This provides a reference to an array object containing each layer as a individual element. Consulting the properties for Layer objects in the Object Model Viewer, we can also discover how to disable a layer using the `.visible` property. Assigning a boolean value to this property will determine whether the corresponding layer is enabled or disabled in Photoshop.

Below is our first attempt at writing the code. We use a definite loop to iterate through each of the layers in the array (see lines 4--7). Then for each layer, we set the `visible` property to `false` in order to disable it (line 6).

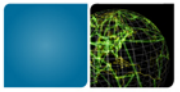
Script Version 1 [hide]

```
1  #target photoshop
2
3  var inputLayers = app.activeDocument.layers;
4  for (var i = 0; i < inputLayers.length; i++)
5  {
6       inputLayers[i].visible = false;
7  }
8
```

Unfortunately testing this script as described in use scenario 1 leaves something to be desired. As you can see from the screenshot link at the right, the program has disable all of the layers, rather than just the text ones. Looking back at the code above it's obvious why this happened. In the body of the for loop we disable each layer (line 6), making no distinction between text and non-text layers. Thus the end result is a document with no visible layers. What we need to add is a way to detect and disable only the text layers.



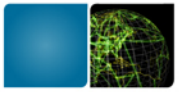
V1 Result


Bottomline: Cases work

- They *like* the cases.
- They *code* the same.
- Case-users *learn* the concepts, while repository users do not.

- Suggests how we might help non-CS professionals who discover computing late.

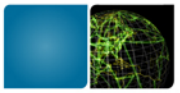




Need 2: Interpreting Code for Search

CS Teaching Need #2:
Given a piece of code, suggest CS concepts and examples, or productive/appropriate search terms.

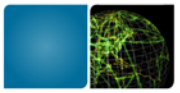
- Patterns to recognize: (simple) recursion, sorting, linked lists, API use.
- Tagging of examples?
- Machine learning approach: Given code X in IDLE or Eclipse, what search terms led to fruitful (defined?) results?

Piece 3: Teaching CS as a study, not just an apprenticeship

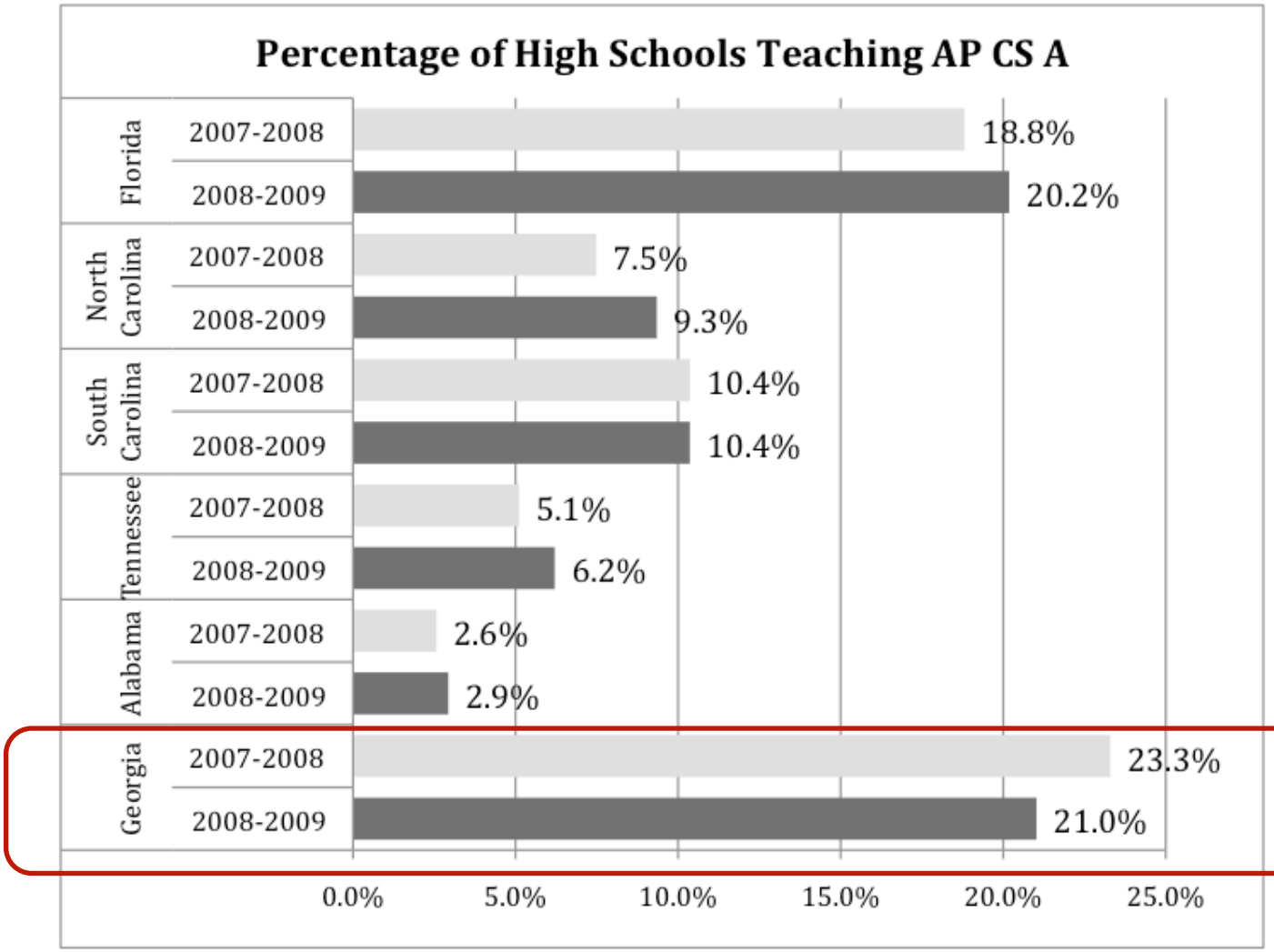

- For the most part, we teach CS as apprenticeship.
  - We model (lecture), students practice (code), we try to coach.
- Practice is *always* important.
  Do we rely on it too much?
  - Are we more like STEM or Architecture?
- Claim: No other science and engineering field teaches so much through practice.
  - Result: CS learning is hard and time-consuming.

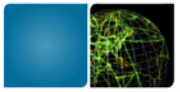
Can we teach more of CS *without* programming?


- Can we teach about variables behavior, how loops work, how conditionals behave – without a half hour of "where goes the semi-colon"?
  - Matt Jadud [2006] (and others) has shown us how small Java errors can lead to an enormous waste of time.
- Can we reduce the wasted time?
  - Analogy: Does coursework in a foreign language make it easier to be immersed in the new language, or is immersion the only way to learn?

Problem: Too little secondary school CS

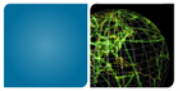



US NSF's Solution: **CS:P => CS10K**

- First, create a new, high-quality course that attracts students and can be taught nationally – that *reduces* the focus on programming.
  - http://www.csprinciples.org

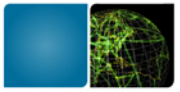
- Secondly, have 10,000 teachers ready and able to teach this course in 10,000 schools by 2015: **CS10K**


How do we go from 2K to 10K?

- *Claim: They are not going to come from pre-service teacher education.*
  - UTeach at U-T@Austin has offered pre-service CS ed for 15 years, with only 7 graduates.
  - Purdue has a program, with only one teacher enrolled.
  - Using pre-service to ramp-up an area doesn't work: The student-teaching problem.
- *Claim: We have to do in-service CS teacher education a whole lot better.*
  - Experiences at Columbus State in Georgia.

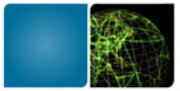
Studying the issues of In-service CS Teacher Education


- Study of adult/professional students in CS classes.
  - They don't have the time to spend hours in front of the IDE.
  - Lacking background, e.g., in mathematics.
  - They get stymied by small errors.

When Life and Learning Do Not Fit: Challenges of Workload and Communication in Introductory Computer Science Online

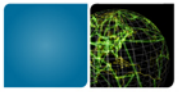
KLARA BENDA, Georgia Institute of Technology
AMY BRUCKMAN, Georgia Institute of Technology
MARK GUZDIAL, Georgia Institute of Technology


"I had my few afternoon hours that I could work on the stuff, but it all just boiled down to me not having time for my family when I was taking the courses. I think the bottom line was with my family structure, I shouldn't have taken more than one course at once." [...] "sometimes I felt like I wasn't putting enough into one class because I was putting so much into the other class." [...] "Then I had to put more time into the family, because I didn't put in as much as I should have, but I still had to put time in for them."

Andrew - "I said one time that I couldn't get this mathematical problem to work. His response was, "I'm not going to teach you algebra." So if you get one little piece or spacing wrong, it doesn't work."

John – "There were times that it would take me hours to find one comma out of place, or find that one something that was wrong, so I didn't mind sticking with it but it just got to the point where I just didn't get it."

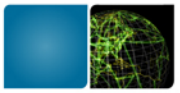

Options?

- If we want to have thousands of high school students studying CS, we have two options:
  - Option #1: Give up on teachers
  - Option #2: Figure out a pedagogy that works for in-service teachers.

Option 1: Don't Use Teachers

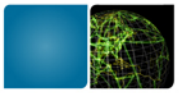
- Dave Patterson (Berkeley) and Alan Kay: *Use Technology Instead*.
- "My belief that the K-12 CS education problem is practically unsolvable for the next 10-20 years in the US is based on:
  1. **No room in the high-school curriculum for CS.**
  2. **Low pay for new teachers.**
  3. **Changing education policy is hard and takes a long time, and there is little reason to believe you will succeed.** This is a state by state, school district by school district level of change involving many advocacy groups.
  4. **Most proposed solutions don't scale.** There are roughly 50,000 high schools and 80,000 elementary schools and middle schools in the US. Whatever you are proposing, think about the time scale your innovation would take to affect 10% of these schools."

Option #2: Figure out a new way to teach high school CS teachers


- Given a focus on in-service teachers:
  - It must be distance education to fit with schedules.
  - Learning must happen *primarily* in 20-60 minute chunks.
- Can learn from Open University successes.
  - But a complete solution does not exist for creating secondary school CS teachers that the US could import.
  - Something we could use for 8K teachers in 4 years?

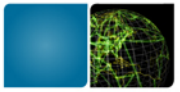
Role 3: Re-Think the Way We Teach CS.

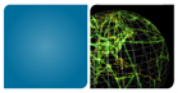

- Could we build technology (e.g., cognitive tutors) to teach *all* of CS:Principles without teachers?
- Or can we build technology to guide teacher learning with *less* programming?
  - Where "book" study can reduce wasted time in front of IDE?
  - Challenge: Not rote memorization, not boring, and usable by teacher with little background knowledge.
  - Idea: A little educational psychology can go a long way.

CS Teaching Need #3:
Create materials that support the learning of computing concepts *reducing* the hours spent programming.

Conclusions


- Computing is important for everyone.
- Students who want computing succeed with contexts.
  - Need #1: Match students to contextualized materials, or develop the ability to tailor materials to a context.
- End-user programmers want what CS has to offer, and there are more of them than there are professional software developers. But they don't know enough CS.
  - Need #2: Support discovery and development of base knowledge.
- Teach CS with less apprenticeship, less programming.
  - Need #3: Provide new approaches to learning CS to support learning of concepts so that time spent programming is fruitful, not wasteful.


With thanks to our supporters

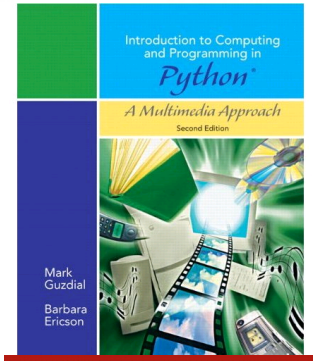
- US National Science Foundation
  - Statewide BPC Alliance: Project “Georgia Computes!” http://www.gacomputes.org
  - CCLI and CPATH Grants
- Microsoft Research
- Georgia Tech's College of Computing
- Georgia’s Department of Education
- GVU Center,
- GT President's Undergraduate Research Award,
- Toyota Foundation

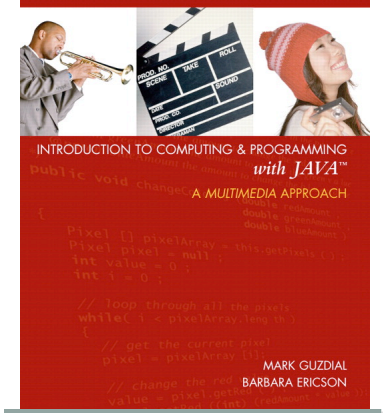
Thank you!

- http://www.cc.gatech.edu/~mark.guzdial

  http://home.cc.gatech.edu/csl

  http://www.georgiacomputes.org



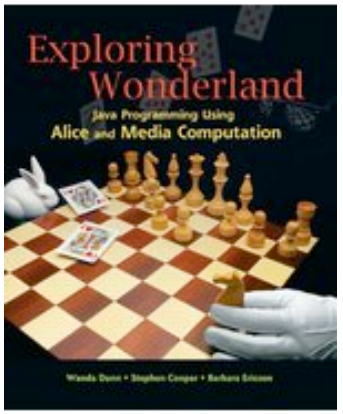


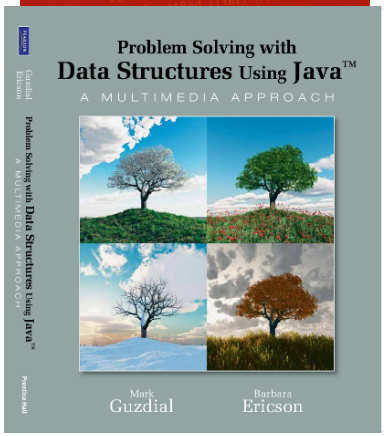
For more on the new APCS.

- http://www.csprinciples.org

For more on MediaComp approach:

- http://www.mediacomputation.org





Georgia Tech College of Computing