

# The First Five Computer Science Principles Pilots: Summary and Comparisons

---

Lawrence Snyder, University of Washington  
Tiffany Barnes, University of North Carolina, Charlotte  
Dan Garcia, University of California, Berkeley  
Jody Paul, Metropolitan State College of Denver  
Beth Simon, University of California, San Diego

## Abstract

In academic year 2010-2011 five professors at different colleges were asked to implement the vision of AP Computer Science Principles. The “vision” was not a packaged curriculum ready to be taught, but rather a *curriculum framework* specified as Computer Science’s Seven Big Ideas and its Six Computational Thinking Practices. The resulting pilot courses reflected a broad interpretation of the 7+6, while still achieving the goals of serving women and under-represented populations. This paper, written by the five instructors, describes those courses.

After a brief overview of the purpose and objectives of teaching pilot courses in the context of the CS Principles project, a summary by each of the teams follows. The summaries are similarly structured to permit easy comparisons across the courses. Finally, a quantitative comparison is presented allowing for further understanding of how the courses connected to the CS Principles content.

## Introduction

In 2010 the Computer Science Principles course existed primarily as two documents, titled, “Seven Big Ideas of Computer Science,” (Table 1) and “Six Computational Thinking Practices” (Table 2). These included bulleted lists of short phrases, which gave the topics and a few key terms for each. What would these topics look like as a college course?

Five instructors from five different colleges were selected to pilot a college course based on the bulleted information during academic year 2010/2011:

- Metropolitan State College of Denver      MSCD   Jody Paul
- University of California, Berkeley          UCB    Dan Garcia & Brian Harvey
- University of California, San Diego        UCSD   Beth Simon
- University of North Carolina at Charlotte   UNCC   Tiffany Barnes
- University of Washington, Seattle          UW     Larry Snyder

<b><i>AP Computer Science Principles: Seven Big Ideas (Fall 2010)</i></b>	
I.	Computing is a creative human activity that engenders innovation and promotes exploration.
II.	Abstraction reduces information and detail to focus on concepts relevant to understanding and solving problems.
III.	Data and information facilitate the creation of knowledge.
IV.	Algorithms are tools for developing and expressing solutions to computational problems.
V.	Programming is a creative process that produces computational artifacts.
VI.	Digital devices, systems, and the networks that interconnect them enable and foster computational approaches to solving problems.
VII.	Computing enables innovation in other fields including science, social science, humanities, arts, medicine, engineering, and business.

Table 1. List of the *Seven Big Ideas*, August 2010, [1]

The schools were the home institutions of five members of the AP Computer Science Advisory Committee. Drawing on Advisory Board members made sense because the instructors had been party to many of the early development discussions, and therefore, had a thorough understanding of the project's goals, emphasis and constraints.

Although the target curriculum was ultimately a high school course, piloting at colleges made sense, because “advanced placement courses” are designed to give college credit.

The pilot instructors had two basic goals.

**Convert Topics to Content.** Because the specification of the Computer Science Principles course took the form of a curriculum framework, an important goal for the pilots was to make the CS Principles content explicit. Though the content represented by the framework's list of topics and objectives and its broad assertions of desired results may have been clear to its creators, it allowed for many interpretations by readers. Illustrating what CS Principles is as classroom content would serve as a tangible interpretation that could be studied and reviewed by a much larger audience.

**Illustrate a “typical” CS Principles Course.** By definition, an AP course teaches content taught in college courses. High school students taking the AP course, and performing well on an AP test, expect to “get credit for” or to “place out of” those college courses. Such courses are not a standardized part of the CS college curricula; see for example ACM's Computer Curriculum 2007 [3]. Further, because computing is taught in several variations – Computer Engineering, IT, IS, etc. – there was no consensus as to what the college target – or targets – should be. The five pilot courses were intended to be exemplars of such college courses.

The course summaries below demonstrate that the five courses are very different from each other. Only two pilots – UCB and UNCC – even attempted to use similar

<b><i>AP Computer Science Principles: Six Computational Thinking Practices (Fall 2010)</i></b>	
1.	Analyzing the effects of computation
2.	Creating computational artifacts
3.	Using abstractions and models
4.	Analyzing problems and artifacts
5.	Communicating processes and results
6.	Working effectively in teams

Table 2. List of the *Six Computational Thinking Practices* [August 2010]

approaches, and although the resulting courses overlap, they were still substantially different courses. Nevertheless, all of the courses taught the CS Principles content. The fact that they are different emphasizes that teaching the fundamental content of computer science can be done in many different ways. This fact also illustrates that campuses differ significantly in how that content fits into their existing course offerings and their campus roles.

### Overview of the Pilots

The five pilots spanned a period beginning in late August, 2010 (MSCD) through May, 2011 (UNCC), with one or more pilots underway during that whole period.

Because the courses were not required to “be the same,” they represent a very interesting combination of approaches. While reading each of them, notice the following characteristics.

**Table 3.** Pilot Overview: “S” is a semester-long course, “Q” is a quarter-long course; “hours” counts total course contact hours; “size” is initial enrollment.

School	Instructor	1st Offering	Title	Hours	Size
MSCD	Paul	Fall 10 S	“Living in a Computing World”	55	20
UCB	Garcia/Harvey	Fall 10 S	“The Beauty & Joy of Computing”	98	80
UCSD	Simon	Fall 10 Q	“Fluency with Information Technology”	50	580
UNCC	Barnes	Spring 11 S	“The Beauty & Joy of Computing”	40	22
UW	Snyder	Winter 11 Q	“Computer Science Principles”	50	22

*MSCD* – The course is very opportunistic and student driven, beginning with student input: What do you want to know? What do you want to learn from this courses? It takes happenstance and current news as additional drivers. It seems that the resulting course would very closely match the student needs.

*UCB* – The pilot was the result of curricular revisions that had already been underway for 18 months. The result benefits from an enormous amount time and development effort, extensive resources, significant contact time, and experienced staff. As a “gateway to the major” course, it has successfully attracted majors.

*UCSD* – By far the largest piloting effort and a degree requirement for much of the class, UCSD’s course applied a Peer Instruction pedagogy as a means of engaging students and giving them responsibility for their learning. Lectures were replaced by student engagement with challenging questions involving peer discussion.

*UNCC* – Drawing heavily on Berkeley’s experience, UNCC’s *Beauty and Joy of Computing* pilot used a structured class period to introduce concepts and apply them immediately in lab activities. This approach helped to accommodate the substantial contact time disparity between UCB and UNCC offerings.

*UW* – Following a dozen years of teaching a related concept-rich introduction (*Fluency*), the *UW* pilot was a new design derived directly from the Seven Big Ideas and Six CT Practices. Relying on a more conventional lecture/demo with structured labs format, students quickly developed teach-it-to-yourself capabilities.

Each contribution provides extensive links to relevant material including the entire corpus of class materials, <http://csprinciples.org>.

## CS Principles Pilot at Metropolitan State College of Denver

---

**Author:** Jody Paul

**Course Name:** Living in a Computing World

**Pilot:** Fall semester, 2010

### a) How *Living in a Computing World* Fit In At MSCD:

Metropolitan State College of Denver (MSCD) is a public, open enrollment, non-residential college attended by 23,000 undergraduate students, 94% of whom are from metropolitan Denver. MSCD has a *statutory mission of providing access for underserved and low-income students*. It is ranked in the top 100 schools in the USA for graduating Latino students and students of color.

Computer Science faculty in the Department of Mathematical and Computer Sciences recognized the opportunity to pilot the course *Living in a Computing World (LiaCW)* as a means to further the objective of making knowledge of computing accessible to all students. The course was open to all with the only requirement being that the students be “college ready”; that is, minimally eligible to enroll in College Algebra 1 and English 1. *LiaCW* did not satisfy any degree requirements other than college-residency units. The course thus had no prerequisite courses nor was it prerequisite to any other course.

### b) Course Specifics

- **Class Sessions:** Two 110-minute periods per week
- **Course:** 15-week semester, yielding 55 contact hours
- **Credit Hours:** 4 semester credit hours
- **Fulfills Requirements:** none (residency only)
- **Attendance:** 20 enrolled; 18 at term’s end
- **Algorithm Specification:** structured English; LightBot; Scratch
- **Grading:** activities, assignments, exams

**c) Class Pedagogy and Content** The class employed an opportunistic approach that emphasized human-to-human interaction.

The order of course content was guided by opportunities arising from the environment of students and their world. On the first day of class, students articulated items of individual and collective interest in response to prompts like “things you’ve wondered about,” “what you would like to know,” and “what you’d like to be able to do.” These formed the surface agenda for the course and the collection was revisited and updated during the semester. In addition, happenstance and current events were considered opportunities to leverage the associated

interest that could facilitate addressing intended content. Such events arose in the news and in the lives of individuals and groups of students.

Class sessions were comprised of shorter sub-sessions during which students primarily interacted with one another toward a purpose. The pedagogical goals were usually to activate a concept, encourage curiosity, and facilitate discovery. Concept activation refers here to engaging in an activity that results in desired information being transferred from long-term memory to consciousness. Such activation and recollection was one foundational element that enabled acquisition of the intended CS Principles content.

Another key element was that of leveraging curiosity in order to encourage the cognitive processing necessary to connect new concepts into a student's existing knowledge framework. The goals were for the activities to arise from the interests of students themselves and to provide both sufficient challenge to be interesting and appropriate challenge to be rewarding. If the challenge was too little, students would be bored; if the challenge was too great, students would be frustrated; either way, learning would be compromised. Achieving the desired balance was difficult even with the small number of students.

Instructor presentations (lectures) were minimized as they were thought to be less engaging and less effective than active-learning experiences. However, the high resource burden necessary to develop active-learning experiences across the entire CS Principles content was too great and this objective was not met to the desired degree. That is, resource cost was a significant impediment to more effective pedagogy.

The course content was drawn from the items articulated in the CS Principles curriculum framework with each of the major areas identified as a big idea receiving roughly the same amount of emphasis. The pilot course environment and timeframe permitted only a subset of the curriculum framework to be addressed.

#### **d) Evidence of Student Work**

The combination of much activity being done by students during class time with the relatively small class size resulted in little difficulty for the instructor and community assistants to directly observe and assess the nature of work and associated learning. In addition, students were required to reflect on each activity and to report insights, observations, and key ideas from each.



*Students created animated dialogs regarding social and ethical issues, such as this image from a clip that highlighted privacy concerns associated with social network postings*

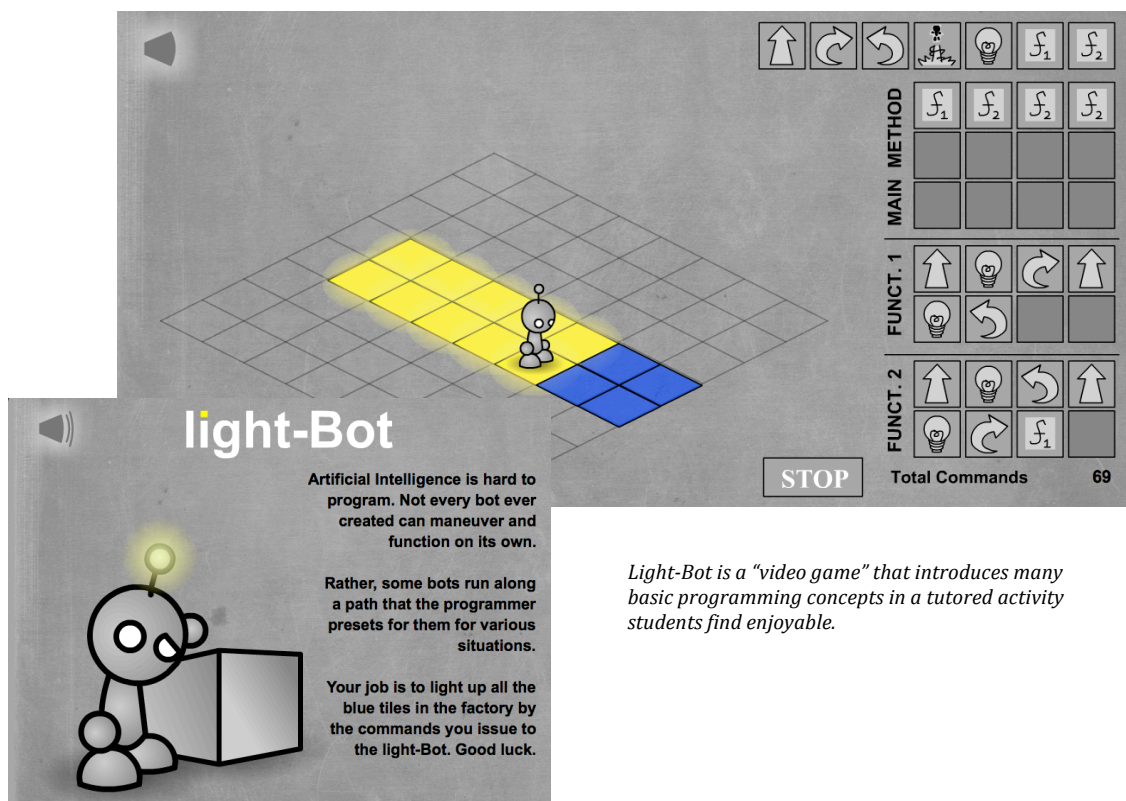
## e) What Worked And Didn't

*Active Learning.* This was, unsurprisingly, a primary contributor to successful learning outcomes. Greatest engagement and retained knowledge appeared to be associated with the active learning experiences.

*Lecture.* Some students found lectures to be the most comfortable, perhaps because lectures were more consistent with their past experiences and expectations and were much less demanding of their attention. Students reported enjoying the presentations, but the associated reflections were generally less focused and personalized.

*Numeric Examples.* This was, without doubt, the most devastating error in the pedagogy. An early appearance of a numerically-based example was sufficient to cause alienation from the concept. This may have been due to the math phobia common among the student population. Far more effective was to introduce and activate a concept using a non-numeric example.

*Lightbot.* Using the Lightbot game proved to be a very effective tool for concept activation, curiosity stimulation, and discovery learning.



**light-Bot**

Artificial Intelligence is hard to program. Not every bot ever created can maneuver and function on its own.

Rather, some bots run along a path that the programmer presets for them for various situations.

Your job is to light up all the blue tiles in the factory by the commands you issue to the light-Bot. Good luck.

STOP Total Commands 69

MAIN METHOD

FUNCT. 1

FUNCT. 2

*Light-Bot is a "video game" that introduces many basic programming concepts in a tutored activity students find enjoyable.*

## AP CS Principles Pilot at University of California, Berkeley

---

**Authors:** Daniel D. Garcia, Brian Harvey, and Luke Segars  
**Course Name:** CS10: The Beauty and Joy of Computing  
**Pilot:** Autumn semester 2010 (and other offerings cited)

### a) How CS10 Fits In At UC Berkeley:

The course is intended for non-CS majors. For students in the College of Letters and Sciences, it fulfills the "Quantitative Reasoning" breadth requirement. It is not required for CS majors, but some intended CS majors with no prior programming experience decide to take it as preparation for our first course for CS majors. In addition, many non-CS majors enjoy CS10 enough to continue with the sequence for majors.



The department, on our recommendation, stopped offering our two other venerable (Scheme programming-only) non-CS major classes so these students would all flow through CS10. We have since seen a dramatic growth in enrollment, from 80 students in both fall 2010 and spring 2011 terms to 240 in both fall 2011 and spring 2012 terms. In the spring of 2011, the course was chosen by the *UC Online Instruction Pilot Program (OIPP)* as an online course pilot, and we hope to go live with the online version of CS10 in the fall of 2012.

### b) Course Specifics

- **Lectures:** Two 50-minute periods per week, MW
- **Labs:** Two 110-minute closed labs driven by Moodle activities with Teaching Assistant (TA) and Lab Assistant supervision, TTh
- **Discussion:** One 50-minute recitation period led by TA (used for lecture review, programming problem work in small teams, reading discussions, CS Unplugged exercises, role playing, computer disassembly, group design of video games, project work, etc.)
- **Format in a given week:** Lecture - Lab - Lecture - Lab - Discussion
- **Course:** 14-week semester, yielding  $7 * 14 = 98$  contact hours
- **Credit Hours:** 4 semester credit hours
- **Fulfills Requirements:** Quantitative Reasoning
- **Attendance:** 80 started; 76 finished
- **Programming Language:** *Scratch* and *Build Your Own Blocks* (BYOB) based on *Scratch*. Soon to be browser-based *Snap!* which works on mobile devices.
- **Grading:** Weekly reading quizzes and homework (15%), 2-3 page paper [later evolving into a 1-page blog with 3 mandatory response paragraphs to other students posts] (15%), Midterm Project (15%), Final Project (15%), Quest [early, sanity-check exam, halfway between a "quiz" and a "test"] (5%), Midterm (15%), and Final (20%).



### c) Course Design

The name of the course originated from Grady Booch's SIGCSE 2007 Keynote in which he exhorted us to share the "Passion, Beauty, Joy and Awe" (PBJA) of computing. We simplified the name (fearing that "awe" may imply more "fear" than "wonder," and that "passion" is internal, not taught) to "Beauty and Joy of Computing" (BJC). Our whimsical logo captures the spirit of the course, while paying homage to its roots (the colors of the BJC letters come from BYOB and the *Blown to Bits* book).

Between the course title, the idea that this may be the last computing course many students ever take, and the need to adequately prepare them for CS61A, several key design principles emerged (many directly from the SIGCSE panels!):

- As much as possible, show beautiful and joyful examples of computing. This includes leveraging (and extending via BYOB) the incredibly well-designed, simple-yet-powerful graphical development environment of Scratch, delightful fractals to teach recursion, elegant higher-order functions that use abstraction to hide messy details of control flow, showing how climate simulations are saving the planet, the internet is wonderfully enabling, etc.
- *Half of the course should be the societal implications of technology.* We assigned an incredible book, *Blown to Bits* (Abelson, Ledeen, and Lewis), which we supplement with videos and other readings.
- Programming should be joyful, so allow students to choose *any project they wish*.
- Deep learning happens by *doing*, not by *listening*, so use the lab-centric model of instruction (here, 4 hours of Moodle-driven labs per week).
- Show relevance of computing to the students. *Every lecture* begins with a "technology in the news" discussion, selected to be most relevant to the student demographic. They are culled from *Slashdot*, *Technology Review*, *NY Times Technology*, *CNN Technology*, *Digg*, etc.
- Lecture should be engaging, so employ *peer instruction* through clickers that are provided as free loaners to all students.
- Invite guest speakers from industry to share "behind the scenes, how the technology behind our business works" talks, to reinforce & ground the importance of the big ideas. We've been fortunate to have accessible and inspiring talks from engineers Raffi Krikorian from Twitter™ and Tao Ye from Pandora™.
- Exciting computing research areas should be presented, by experts in that area. These should be a broad "here's the entire field" overview, not a narrow "here's my research" talk. We invited departmental faculty Bjoern

Hartmann and Armando Fox to speak on HCI and Cloud Computing, and course instructors added their survey of the field of artificial intelligence.

- Make the entire course free to students, so the class can be exported to high schools easily. Everything from the IDE to the book is available for free.
- Learning to program can be deeply frustrating, so encourage both their three-week projects to be done in pairs (even the occasional teams of three).
- It sometimes takes a while for this material to “click” and students often have a bad day, so allow a high score on a later exam to overwrite a poor performance on an earlier exam.

### Lecture Topics (and number of lectures on that topic)

- **Abstraction** (1) – Abstraction as a reasoning and problem solving tool
- **Video Games** (1) – The development of video game platforms and technology, games with a purpose (GWAP)
- **3D Graphics** (1) – Fundamentals of 3D graphics (a fun, “How It Works” explanation)
- **Algorithms** (2) – Divide and conquer algorithms and analysis techniques
- **Functions** (1) – How to create functions in BYOB and the beautiful implications
- **Programming Paradigms** (1) – The benefits and trade-offs of each
- **Concurrency** (1) – Dealing with concurrency on one computer, both historical and BYOB-based
- **Distributed Computing** (1) – Dealing with concurrency between computers, both historical and BYOB-based
- **Recursion** (3) – Emphasis on graphical recursion (fractals) and non-linear recursion since looping constructs are available
- **Social implications of computing** (2) – Discussion of the ethical, economic, security and privacy implications of computing (Education, privacy, risks, Patents and copyrights, war, community, etc.)
- **Applications that changed the world** (1) – Commentary and technical details of some applications of computing that have changed the world (Transistor, PC, WWW, Internet, Search Engines, Google Maps, Video Conferencing, Social Computing, Cloud Computing, etc.)
- **Computing in Industry** (1) – Industrial guest (Twitter, in Fall 2011 we added Pandora) explains how they use computing
- **Saving the World with Computing (CS + X)** (1) – Guest from Lawrence Berkeley National Labs talks about climate simulation, protein folding, digital humans, and other supercomputer activities that are saving the world

- **Higher Order Functions and Lambda** (2) – Higher order functions and lambda. The same programming ideas introduced in CS3 (the beauty and power of functions as data, anonymous functions)
- **Cloud Computing** (1) – Fundamentals of cloud computing (a “How it Works” explanation)
- **Game Theory** (1) – Deep Blue and general game strong solving techniques
- **Artificial Intelligence** (1) – Historical progress and current directions and innovation in AI
- **HCI** (1) – Historical progress and current directions and innovation in Human-Computer Interfaces
- **Limits of Computing** (1) – Revisit analysis techniques and discuss NP problems, computers are finite (for representations)
- **Future of Computing** (1) – Discussion of current and future areas of innovation and research
- **Wrap-up** (1) – Project demos and a sneak peak at EECS undergrad course offerings

#### **Lab Topics (and number of labs on that topic)**

- **Learn Scratch** (3)
- **Learn BYOB** (1) – Learn how to create “functions” or “Blocks” in BYOB
- **Lists** (2)
- **Algorithms** (2)
- **Concurrency** (1)
- **Distributed Computing** (1)
- **Recursion** (3)
- **Higher Order Functions and Lambda** (2)
- **Hash Table** (1)
- **Simulations** (1)
- **Work on projects** (6)

**Reading List:** See the course web site (below) for a complete list.

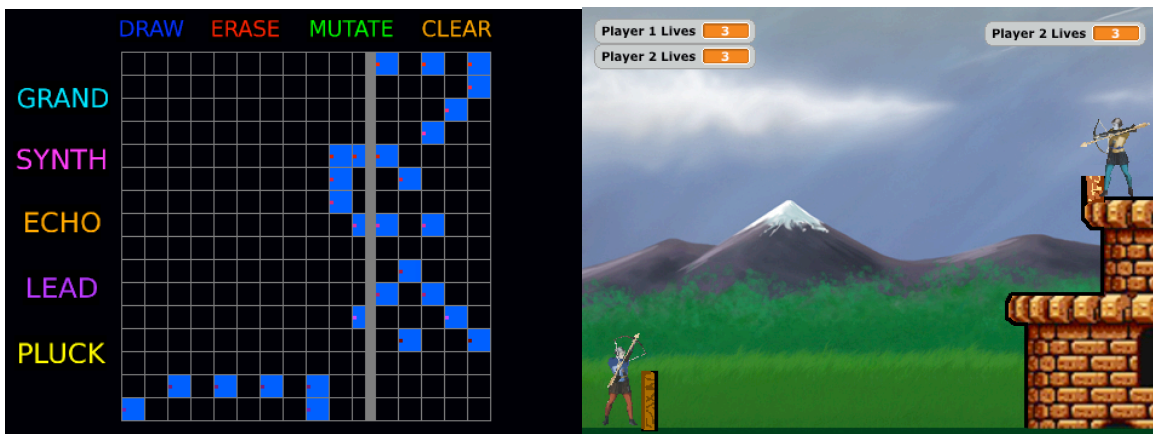
#### **d) Evidence of Student Work**

Student papers were topical and interesting. Samples titles: Net neutrality, Prosthetic Arms, Record Companies No Longer Rule the Roost, Computers in Astrophysics, News for the Information Age, AI & Transportation and Electronic Privacy.

Three programming projects (across three consecutive years of teaching the course) were chosen as the best that students ever produced (in only three weeks):

*Picoboard Guitar Hero* (Pierce Vollucci and Sina Saeidi, fall 2009); these students used a USB PicoBoard sensor board and a keyboard and built their own guitar, then wrote a *Guitar Hero*-style game to train them to play it! The musician would press keyboard keys for different notes and pluck one of the wire frets, which would complete a circuit, and the PicoBoard would let BYOB know what note to play. They performed “The Battle Hymn of the Republic” during the final presentations!

*Magic Machine 3.0* (Max Dougherty and Victor Lyamar, fall 2010); this incredible program featured an editable 2D grid; time vs instrument (with higher-pitched instruments at the top, drums at the bottom). Clicking in a grid location placed a blue square there, meaning that note should be played at that time. A grey vertical bar slid to the right (and reset on the left after it reached the end), and played the multi-track loop over and over. The user could change the instrument using the words on the left column and change editing modes using the words on the top.



*Kinect Archery* (Jim Knudstrup, Amanda Atkinson and Vivian Lo, fall 2011); these students hooked up a Microsoft Kinect to BYOB, and authored a very playable archery game that used the Kinect to aim the bow, and physics to control the trajectory of the arrows. Video available at <http://youtu.be/wlHwbCcujo>

### e) What Worked And Didn't

*Sufficient development time, outstanding team, small half pilot ... worked.* The course was developed over the eighteen months prior to the official AP CS pilot by three teaching-track faculty and ten of the strongest graduate and undergraduate teachers. By the time we launched with the full course pilot in the fall of 2010, we felt we had debugged most of the issues with the software and course content.

*Peer Instruction ... worked.* Students responded that they were delighted they didn't need to pay for clickers (as in all their other classes) and that it made lecture more interesting and dynamic.

*Videotaping lectures ... mostly worked.* We learned that it's hard to have a well-lit instructor as well as a dark screen, and to keep lecturers from pacing the floor.

*BYOB ... mostly worked.* Several key bugs were squashed during the fall 2009 "half pilot" as development continued. The from-the-ground rewrite now in progress (called *Snap!*) will address the remaining speed problems and will run in a browser and on every smart mobile device.

*2-3 Page Paper ... didn't scale, and didn't take advantage of the social component.* The care and effort we gave to the papers for the half-pilot (each of the co-instructors read every paper) clearly didn't scale as we moved to 80 students for the pilot. We replaced this assignment with a one-page blog where students were required to read three other students' blogs (two assigned by us, the other of their choice) and write at least a single paragraph of reflected commentary for each.

## **f) Links, Resources, Acknowledgments**

The fall 2010 pilot was co-taught by Dan Garcia and Brian Harvey, with help from teaching assistants Luke Segars and Jon Kotker. The fall 2009 pre-AP course was co-developed by Dan, Brian, Jon, Mike Clancy, Colleen Lewis, George Wang, Glenn Sugden, Stephanie Chou, Brandon Young, Wayland Siao, Gideon Chia, and Daisy Zhou, with collaboration of local high school teachers Ray Pedersen, Eugene Lemon and Josh Paley.

The entire course curriculum is available at <http://bjc.berkeley.edu>. We offer teacher preparation workshops to groups of 20 or more teachers in a locality; participants get a stipend and we bring an experienced BJC instructor to your site. See the web site for more details.

### **The current class Web Site:**

<http://inst.eecs.berkeley.edu/~cs10/>

### **Build Your Own Blocks (BYOB) based on Scratch, and Snap!**

<http://byob.berkeley.edu>

### **Testimonials about CS10 : The Beauty and Joy of Computing**

<http://www.youtube.com/playlist?list=PL48C2AF5762B2D32D>

### **Fall 2010 High-Definition lecture videos:**

<http://www.youtube.com/playlist?list=PLECBD29A17AAF6EF9>

### **ensemble Computing Portal for commenting and rating BJC learning items**

[http://www.computingportal.org/bjc\\_collection](http://www.computingportal.org/bjc_collection)

## AP CS Principles Pilot at University of California, San Diego

---

**Authors:** Beth Simon (UCSD) and Quintin Cutts (University of Glasgow)

**Course Name:** CSE3 Fluency with Information Technology

**Pilot:** Fall 2010 quarter and Winter 2011 quarter

### a) How CSE3 Fits In At UCSD:

UCSD's Computer Science and Engineering department has offered CSE3, *Fluency with Information Technology* for a decade. In the past 5 years or so, it has been a required course for two groups on campus: psychology majors and students enrolled in UCSD's Sixth College undergraduate program. At UCSD all students have their general education requirements set through their choice of a "college"; and Sixth College's focus is on culture, art, and technology. The Pilot offerings in Fall 2010 and Winter 2011 served ~1000 students. Sixth students (primarily freshmen) accounted for 70% of the population and were spread across all majors except engineering. Psychology majors were primarily juniors or seniors. The course was ~60% women. The course content of CSE3 had been under evaluation for a year when Dr. Simon was presented with the opportunity to pilot CS Principles. Upon review, the newly recommended CSE3 curriculum appeared to fulfill the grand majority of CS Principles learning goals and, hence, seemed a great fit.

### b) Course Stats

- **Lectures:** 2 80-minute periods, TTh
- **Labs:** 1 120-minute closed lab with TA instruction, spread over the week
- **Course:** 10-week quarter, yielding 50 contact hours
- **Credit Hours:** 4
- **Fulfills Requirements:** General Ed for Sixth College, Psychology Major
- **Attendance:** Fall 2010: 580 started; 572 finished. Winter 2011: 456 started; 447 finished
- **Programming Language:** Alice followed by Excel
- **Grading:** daily quizzes, daily peer instruction/clicker questions, 9 lab exercises, individual programming project, midterm, final
- **Pilot:** Fall quarter, 2010, Winter quarter 2011

### c) Course Design

The course was designed with the following key underlying questions: If this were the last course someone was to take in computing, what would you want them to know? What do you want them to get out of it? Our answer to this was to help students develop not only the knowledge, but the visceral understanding that computers are:

1. Deterministic – they do what you tell them to do
2. Precise – they do *exactly* what you tell them to do, and

3. Comprehensible – the operation of computers can be analyzed and understood.

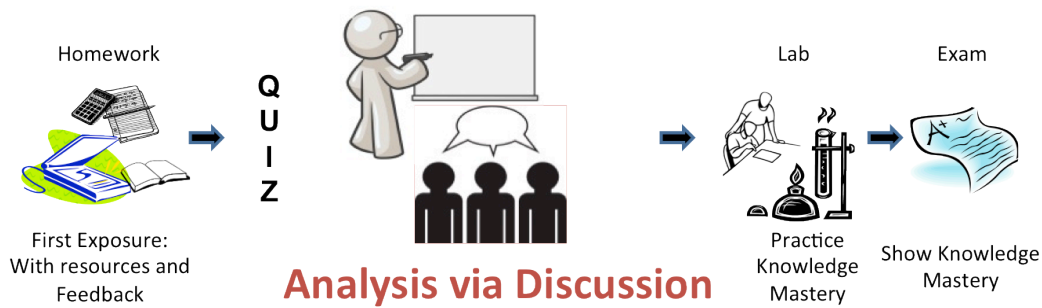
As such, we agreed with the CS Principles belief that “programming” is one of the best tools we have for helping students get at the *heart* of computing – that you can provide directions to the computer and it will do what you say. But our previous experiences led us to believe that simply “getting students to program” didn’t effectively convey that message to the masses. We’d seen all too many students leave introductory programming experiences not with a sense of control, but with a more firmly developed sense of mystery, that computers can’t be understood and you just move things around and change stuff until things finally do what you want.

We took as our mantra to students that we didn’t care so much if they could write a program to do something – if they couldn’t also explain how it worked and be able to analyze what it did and why. Analysis and communication – not program writing – was our goal for students. To support them in developing these skills, we adopted course design based in the Peer Instruction pedagogy [4]. In a nutshell, Peer Instruction recognizes that it is students who need to do the learning (not the professor) and provides the structure and time for students to spend in analysis and discussion of challenging questions that engage them in confronting difficult concepts. The standard course “lecture” is replaced by a two-part process:

- 1) Preparatory work outside of lecture (pre-reading of the textbook, though in this class augmented by work in Alice building the programs described in the text) and
- 2) Replacement of standard “demoing” or explanations with a series of multiple-choice questions where students are engaged in analyzing a problem or some code, and describing what the code does or selecting some code to make a specific thing happen.

The preparatory work involving program building derives from our awareness of students’ need for real experience of programming concepts and constructs if they are going to have meaningful discussions about what programs mean. Simply reading about programming doesn’t give the necessary depth of experience or engagement – learners need to actually see the constructs working.

The in-class multiple-choice questions are asked using a specific format. First, each student attempts the question him/her-self (and indicates their answer electronically with a clicker). Then, students spend a few minutes discussing the question with a pre-assigned group of 3 students. The groups are exhorted not just to agree on the answer they think is correct, but to also explain their thinking to each other and to discuss why the wrong answers are wrong. Finally, a second round of answers is collected by clicker, and a class-wide discussion is led where students are asked to share the thinking and ideas discussed in their groups. The instructor can augment these explanations by modeling how she thinks about the problem and clarifying any specific misunderstandings that might persist.



*Schematic diagram of the peer-teaching approach.*

Final assessment of understanding is measured through weekly 2-hour closed labs (covering the previous week’s materials) and standard exams (mostly multiple-choice questions, with some written explanations and code writing required).

In addition, students completed both a 4-week individual project in Alice to create a digital animation or video game of social value (communicating their views on a subject of import to society, or providing educational value). The course goals were also supported by three “Technology and Society” assignments that sought to focus students on how the programming concepts they were learning existed in the world around them. These also engaged students in digital communication techniques such as discussion forum posting and wiki creation and editing. The way in which these additional assignments were able to link the students’ programming work with their everyday lives sought to give *meaning* to the whole process.

The course content was derived from *Learning to Program in Alice* (Dann, Cooper, Pausch) Chapters 1,2 (sequential execution), 4 (methods, parameters), 5 (events), 6 (functions and if statements), 7 (loops), and 9 (arrays/lists); and a popular Excel book (basic formulas and functions, managing large datasets, exploring data with PivotTables).

#### **d) Evidence of Student Success/Valuation**

As a general education course, one important metric of success is pass rate for the course. Yes, this course needs to be rigorous and provide students the training and understanding to be confident and capable in using technology in their lives. However, students also need to be able to succeed in the course without lengthening their time to degree. Given high failure rates in standard introductory computing courses, this is a non-trivial issue. The pass rate for the fall term was 98% and the pass rate for the winter term was 97%.

But did we succeed in meeting our educational goals? To help measure this, we sought answers to questions like: “What if this is the last computing course these students ever take? What are they getting out of it? Does this satisfy us with regards to what an informed populace should know?” We were teaching “programming” but our lectures focused students on analyzing programs to illuminate core concepts and skills. We asked students about their experience in lab during Week 8:



*Learning computing concepts may have opened many doors for you in your future work. Although you may not ever use Alice again, some of the concepts you have learned may become useful to you. Some examples include:*

- *Understanding that software applications sometimes don't do what you expect, and being able to figure out how to make it do what you want.*
- *Being able to simulate large data sets to gain a deeper understanding of the effects of the data.*
- *Understanding how software works and being able to learn any new software application with more ease, i.e. Photoshop, Office, MovieMaker, etc.*

*Aside from the examples given, or enhancing the examples given, please describe a situation in which you think the computing concepts you have learned will help you in the future.*

A more complete analysis of students' answers can be found in [5], however we found seven categories of response types among students' responses, with students reporting on average at least two of these each:

- Transfer, near: can apply new skills in software use
- Transfer, far: can use problem solving skills in other areas of life
- Personal Problem Solving Ability: Debugging – can logic it out, attempt to, or deal with, unexpected behavior
- Personal Problem Solving Ability: Problem Design – can develop plan to solve technical problem, can see what requirements exist
- View of Technology: greater appreciation or understand of technology
- Confidence: increased ability to do things on computer, a can-do attitude
- Communication: communicate better about technology

The student answers themselves convinced us that the class had had a notable impact on many students' lives. As an example, one student answered:

*"The things I learned in Alice can help me not to be so frightened in general when dealing with technology. Although I am not certain I have absolutely mastered every concept in Alice, I am certain that I have learned enough to bring me confidence to apply these ideas in the technological world. This is a big deal for me, as I do consider myself quite technologically challenged. I think this class has given me tools for life, that can be applied to both my life at home, socially, and at work."*

#### **e) What Worked And Didn't**

*The Language.* Worked. The drag-and-drop nature of Alice did provide students the reduced frustration and cognitive load in developing programs – enough so that we were able to ask students to “develop” programs on their pre-class preparation, without them getting stuck half-way. However, just as important was the fact that the execution model of an Alice program is “visible” to students – that is, for every instruction they create they can “see” the effect of that instruction in execution of

their program. (Note: we didn't cover variables in our seven weeks of instruction – somewhat a shock to us as instructors initially, but, in the end, a critical recommendation from Steve Cooper). In group discussions in class it was clear how students would match up observed behavior and instructions – and how they came to understand that debugging is a comprehensible process of matching code with outcomes.

*Peer Instruction. Worked.* Peer Instruction's core premise is to provide students the opportunity and excuse to engage with deep challenging concepts. Originally developed in physics to help address the fact that students could “plug and chug” their way to correct answers without really understanding applicable physical laws, Peer Instruction was an excellent fit for the goals of this course. Programming is a great tool for engaging students, because they can see for themselves if they have “got it right”. However, that's also a potential pitfall. We used Peer Instruction to keep students focused on the goal of understanding how programs and programming concepts worked – not just getting a program to do what you want.

*Excel. Worked.* It was surprisingly rewarding to cover “basic” Excel topics *after* having done Alice. Students were somewhat amazed to see how previous experiences with Excel could now be seen in a new light. Students saw how a function like sum was, well, a function. It takes parameters and returns a number value. Nested if statements in Excel were equally easy. Students could identify (on the exam) that VLOOKUP is a loop over an array of items with an if-statement inside.

*Focus on Analysis and Communication. Needed improvement.* The first term of the pilot we did not make clear enough to students (from the beginning) that our goal was to help them develop technical analysis and communication skills, both to be able to interact with computers better themselves, and to be able to interact more effectively with computing professionals. Students are spending a lot of time programming in Alice, and it's important to emphasize almost daily how learning to use Alice is just a means to an end – the end being an understanding that computers are deterministic, precise, and comprehensible.

*Multiple Classrooms via Video. Didn't work.* For reasons outside the scope of this pilot, each term the class was offered simultaneously in three rooms co-located in one building. The instructor “lectured” in one room and video of her was streamed live to the adjoining rooms. Although all rooms were staffed with actively engaged tutors and TAs (e.g. as support during discussion periods) and that the instructor walked among all the rooms during most every discussion period, students made it clear that they didn't like this “video” format – though the concern was primarily among freshmen; upper division students felt that the style of the classroom (using Peer Instruction) was all that really mattered.

## f) [Links, Resources, Acknowledgments](#)

The pilot was taught by Beth Simon, but collaboratively developed and inestimably improved by Quintin Cutts, visiting on sabbatical from the University of Glasgow.

Extreme thanks go to our intrepid army of TAs and undergraduate tutors (all 40+ of them), without whom the course could not have been the same.

A web site for educators interested in adopting our curriculum:

<http://www.ce21sandiego.org/>

## AP CS Principles Pilot at University of North Carolina at Charlotte

---

**Author:** Tiffany Barnes

**Course Name:** The Beauty and Joy of Computing (course number pending)

**Pilot:** Spring semester, 2011

### a) How BJC Fits In At UNC Charlotte:

UNC Charlotte's Computer Science department offers ITCS1212, *Introduction to Computer Science*, a course with an accompanying lab, which is taught to 300 students (typically freshmen) from across campus each semester. It is a required course for a few departments. ITCS1212 is consistently over-subscribed and further expansion is resource limited. This course is the first in a series of courses for CS and IT majors, and teaches basic programming concepts in C++. Non-majors find this course difficult and leave it unsure about how it connects to any other subjects.

ITCS 1203 *Beauty and Joy of Computing* is a preliminary course, cross-listed for Computer Science and Information Science, intended to introduce non-majors and majors with little to no computing experience to the general concepts of computing. *Beauty and Joy of Computing* stresses the connection between computing and other disciplines while building a strong foundation in general computing concepts through hands-on visual and interactive projects.

### b) Course Stats

- **Lectures:** 2 75-minute periods, TTh
- **Labs:** Built into the lecture schedule, about ½ of class time
- **Course:** 16-week semester, yielding 40 contact hours
- **Credit Hours:** 3
- **Fulfills Requirements:**
  - Computer Science General Elective (Spring 2011/Fall 2011)
  - **Future:** Hopefully General Ed, Mathematical and Logical Reasoning
- **Attendance:** 22 started; 20 finished
- **Programming Language:** SNAP! (BYOB, a version of Scratch), AppInventor, GameMaker, and StarLogo TNG
- **Grading:** 21 homework assignments, 13 labs, midterm, final

### c) Class Content

The class followed the schedule shown below.

Classes usually consisted of an introduction to a topic via Powerpoint slides and videos, a CS Unplugged style class activity or a discussion, and a lab. Class topics were chosen to be relevant and to illustrate computing's connection to society and innovation. Labs gave students hands-on experience with the material. Most of the lab exercises were done in pairs to be completed as homework. Students were thus motivated to complete as much of the labs in class as possible. The intention was for each lab to be doable by students in under 2 additional hours outside of class.

Weekly readings emphasized the impacts of computing, both positive and negative, on society, and were usually discussed through online forum discussions.

The class focused on making computing relevant while also demonstrating fundamental computing concepts in a hands-on interactive way. Guest lectures from departmental faculty and a doctoral student emphasized computing research topics and how they impact society.

- Richard Souvenir – Computer Vision (classifying what’s in a picture/video)
- Jamie Payton – Participatory sensing (incentivizing crowdsourced data collection)
- Lane Harrison – Visualization (how it can help people solve problems)
- IBM Watson – Campus-wide presentation by a Project manager at IBM
- UNC Charlotte research overview; opportunities for undergraduate research

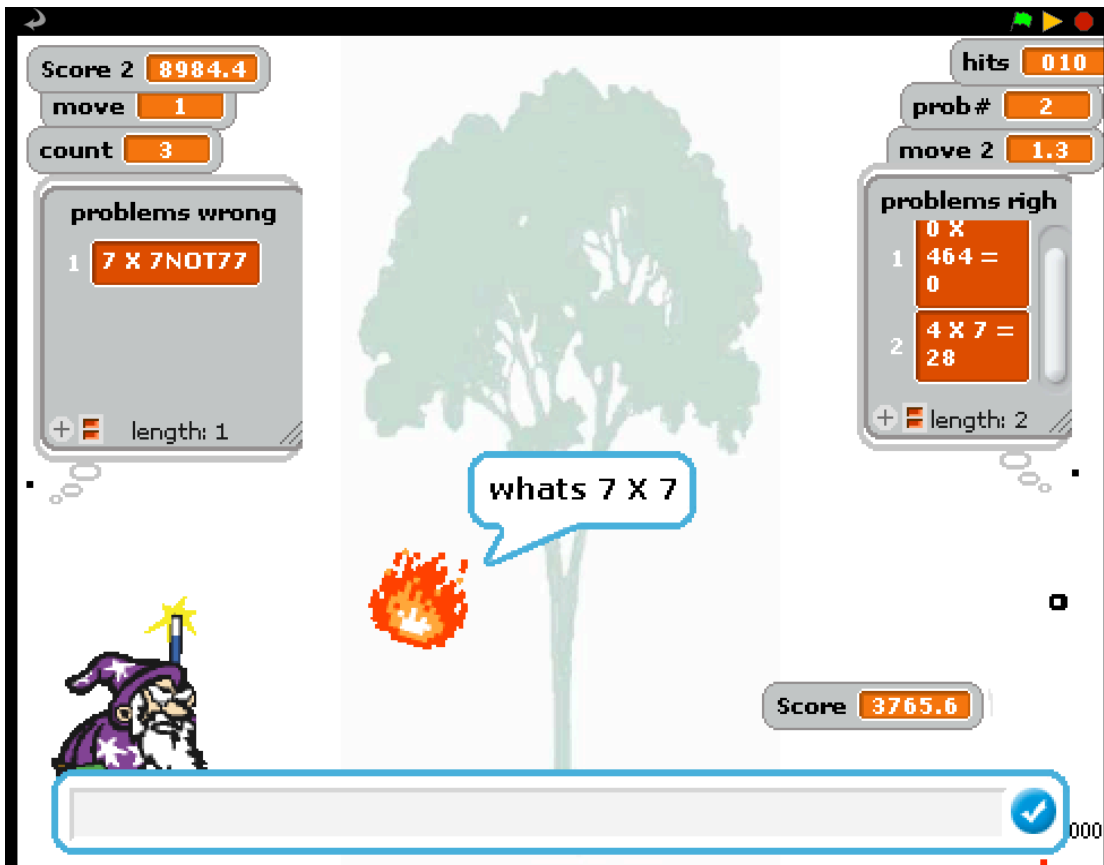
Students reported that the class filled gaps they had in computing knowledge, even for those who took the course as an elective in their senior year in computing. Two students applied for and participated in a summer REU after the course; one of these was a business major who is changing his major to IT.

#### Topic List for UNC Charlotte’s Beauty and Joy of Computing course

1. Welcome & Orientation
2. Scratch Intro lab
3. Scratch Controls & Variables lab
4. Video Games lecture
5. Gamemaker Intro lab
6. BYOB – make your own blocks lab
7. OO Programming; Controls lab
8. Algorithms lecture; Lists Intro lab
9. Complexity; Using Lists lab
10. Artificial Intelligence - Videos
11. Search and Sort Algorithms
12. Recursion Unplugged activity
13. Recursion; Fractals lab
14. Midterm review
15. Midterm
16. Project Introduction
17. Project Proposal Presentations
18. Applications that Changed the world – demos; discussion
19. Computer Vision & Visualization guest lectures
20. Research opportunities & Visualization guest lecture
21. HTML and usability
22. Project work (lab)
23. Usability testing of projects
24. Project presentations
25. AppInventor lab
26. AppInventor lab 2
27. Simulation BYOB lab
28. StarLogo TNG simulation lab
29. Higher Order Functions lab
30. Final exam (includes societal impact essay)

#### d) Evidence of Student Work

Of particular importance in our class is promoting students to use computing for their own goals. Projects ranged from a math multiplication game, to a Jeopardy! Interface, to a colonial times history lesson, to a tournament scheduler, to a memory and math game that was actually used for kindergarteners.



Water Bender was a favorite project in the class. It's a math practice game, where you solve randomly generated multiplication problems to save a tree; solving one problem douses one spark that can ignite the tree. This program was quite challenging for the students to design – the sparks travel toward the tree, and speed up as time goes on, becoming faster as the player misses more problems. The impressive thing about this game was that it was tuned to match with the player's ability. This is difficult to accomplish in any game, let alone one made in 4 weeks by novices.

#### e) What Worked And Didn't

*Scratch/BYOB/SNAP!* This easy-entry programming environment based on one made for children made it easy for novices to learn to program. However, we did jump right in and scared a few might-have-taken-this-class students away with labs to complete at home. The extension of Scratch that allows you to make your own

blocks, and those that allow for higher-order functions, make it easy to teach advanced concepts in an early course.

*GameMaker, AppInventor.* Students were very excited to learn to make games and Android phone applications. GameMaker was easier for programming large numbers of objects, and made a great contrast for talking about programming paradigms, since it is more strongly object-oriented than Scratch/BYOB/SNAP! App Inventor helped students feel that they could make cutting edge programs that other people might actually use. Since it uses the Scratch programming interface, learning it was easy.

*StarLogo TNG:* This environment felt clunky compared to the rest, and since it ran slowly on our old laptops, this feeling was magnified. Some students liked the simulation aspect but we felt we could have found or created labs that made a better connection to biology and student's lives or at least other subjects.

*HTML.* We did one class and homework on HTML but it was much less exciting than other topics. Many students had already looked at HTML before; we felt we could have skipped this topic since students can learn it easily on their own.

*Abstraction.* It's the #2 idea in the "Big Ideas" list, and it was a topic addressed most weeks. We felt this was a success – students did seem to be working at the right level of abstraction on most projects. This might not be as explicit as it could be.

*Sequencing.* Student feedback showed appreciation for learning multiple tools but wished for them to be "chunked" so all BYOB was together and the other applications came later. This was done in Fall 2011 and worked well.

*Projects.* Allowing students to choose their own projects and teams was a great success. We initially planned for 2 projects but reduced it to one after we saw the high quality of those turned in. In Fall 2011, we went back to 2 projects but the second was a small one to demonstrate deeper understanding of one newly introduced tool (GameMaker, AppInventor, or StarLogo TNG). This allowed students to spend more time in one tool they enjoyed. Usability testing in class really drove home the importance of testing code and just how difficult it is to get that final level of polish to a piece of software.

*Readings.* Understanding of the readings would be better if they were explicitly discussed in class or if there were peer reviews of one another's posts in the forums.

## **f) Links, Resources, Acknowledgments**

UNC Charlotte's Beauty and Joy of Computing class was adapted from Dan Garcia and Brian Harvey's BJC class at Berkeley. The UNC Charlotte class is distinct in that it has less than half of the contact hours of the Berkeley course, but supplements the course with learning multiple programming environments and tools. This is accomplished via fewer and shorter lectures, lab time in class, and discussion of readings done online.

The pilot teaching assistants were Drew Hicks (then and now a doctoral student and now NSF Graduate Research Fellow) and Shaun Pickford (then undergraduate, now graduated with a job at Microsoft). Drew and Shaun helped extensively in preparing course lectures and CS Unplugged activities.

Four high school teachers in Charlotte including Sharon Jones, Beth Frierson, Renada Poteat, and Brian Nivens, observed some days of the course, worked through labs, and designed assessment questions for adoption when the course is offered in high schools.

- The class Moodle (login as guest): <http://moodle.game2learn.com/>
- The BYOB/SNAP! Page <http://byob.berkeley.edu/>
- The Blown to Bits textbook: <http://www.bitsbook.com/>
- GameMaker: free download available at <http://yoyogames.com>
- App Inventor: <http://appinventoredu.mit.edu/>
- StarLogo TNG: <http://education.mit.edu/projects/starlogo-tng>
- The Beauty and Joy of Computing “Frabjous CS” project site for Dan Garcia, Brian Harvey, and Tiffany Barnes project to train 100 HS teachers to teach BJC: <http://bjc.berkeley.edu>



## AP CS Principles Pilot at University of Washington

---

**Author:** Lawrence Snyder

**Course Name:** CSE120 Computer Science Principles

**Pilot:** Winter Quarter 2011

### a) How CSE120 Fits In At UW:

UW's Computer Science and Engineering department has offered CSE100, *Fluency with Information Technology* for a decade in collaboration with the University's Information School. That class is taught to 150 students (typically freshmen) per quarter from across campus. It is a required course for a few departments. FIT100 is consistently over-subscribed and further expansion is resource limited. The course is general interest, and is considered preliminary to the standard introductory sequence, *Computer Programming I & 2*.

CSE120 *Computer Science Principles* was added as a new preliminary class. It is distinguished from *Fluency* as follows: Both classes are concepts classes that teach similar content: *Fluency* emphasizes concepts to make students more facile with computation; *Principles* emphasizes concepts as science and stresses the clever and amazing ideas of CS as a science.

### b) Course Stats

- **Lectures:** 3 50-minute periods, MWF
- **Labs:** 2 50-minute closed labs with TA instruction, TTh
- **Course:** 10-week quarter, yielding 50 contact hours
- **Credit Hours:** 5
- **Fulfills Requirements:** General Ed, Quantitative & Symbolic Reasoning
- **Attendance:** 22 started; 22 finished
- **Programming Language:** Processing and XML with some HTML/CSS
- **Grading:** 20 homework assignments, 5 lab exercises, midterm, final

### c) Class Content

The class followed the schedule shown below.

Lectures were a combination of conventional Powerpoint slides, online demos and discussion. Labs were generally designed to give students hands-on experience with material illustrating or implementing concepts discussed in the preceding lecture. Some of the lab exercises were graded.

In general a homework assignment would be given at each lecture to be turned in prior to the next lecture. The exercises were varied, and doable within two hours by a typical student. Students were asked to spend one hour per day engaged in class work, as opposed to two hours in one sitting. Eventually the assignments became larger and spanned several classes. Lab time was allocated so the TA could answer

questions. Though students preferred the less frequent assignments, at the start the relentless every-other-day schedule brought everyone quickly up to speed with basic concepts, esp. programming ideas.

The class emphasized using CS Principles in other majors, rather than trying to motivate students to major in CS. (UW's program has severely limited enrollment and accepts far fewer than the number of qualified students.) Two guest lectures from departmental faculty emphasized the use of fundamental CS for non-technical purposes.

- Richard Ladner: Accessibility – Using Computing To Help The Handicapped
- Zoran Popovic: FoldIt and Other Games To Make The World Better

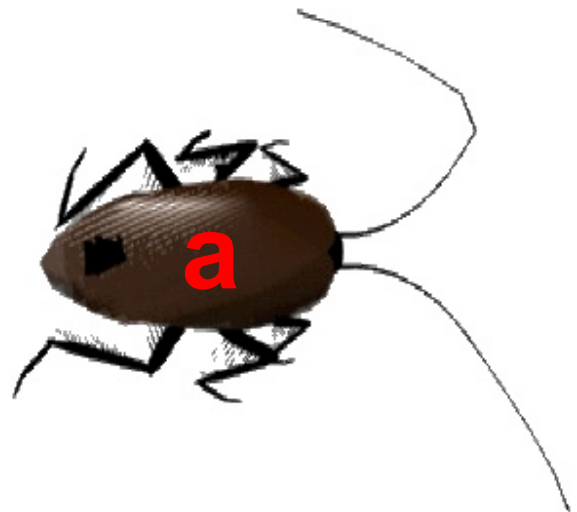
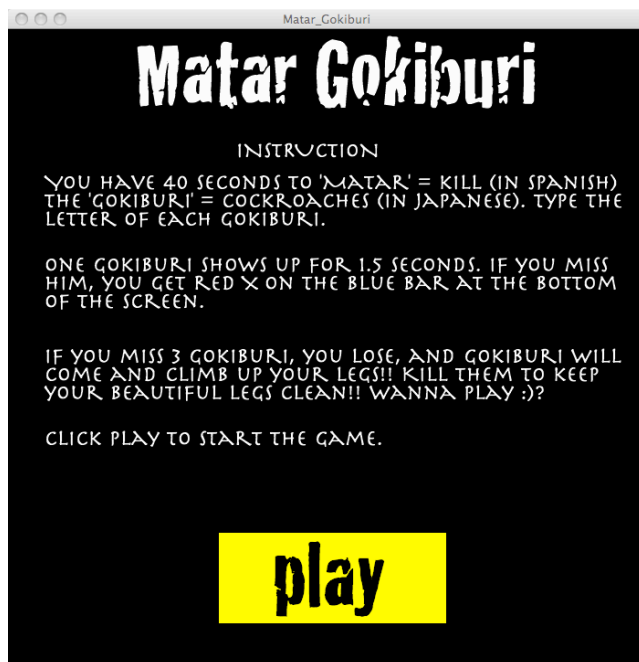
Students found both lectures extremely inspirational. Regarding majoring in CS specifically, there was an “Ask The Advisor” reception (with pizza) before the first Watson Jeopardy Match. Several students have started the path to be CS majors.

#### Topic List for UW's CS Principles in Time Sequence

1. 1 Welcome & Orientation
2. “What I Value” writing exer. (lab)
3. Lightbot: Game or Programming?
4. FTP Review & Portfolio Page (lab)
5. Digitization: History of 0 & 1
6. Computing in Social Setting
7. Bits: Counting in bases 2, 16 (lab)
8. Basic Processing Ideas
9. Q & A on Processing (lab)
10. Digital Representation
11. MLK Holiday – no class
12. Writing Processing Pgms (lab)
13. Pac-Man – Abstract in Proc'g
14. Populating Portfolio Page (lab)
15. If's and For's in Processing
16. Color, RGB, binary arithmetic
17. Practice w/binary adding (lab)
18. Fetch/Executing – How it works
19. Work on Own Programs (lab)
20. Sports Timer: Abstraction Layers
21. Digital, Analog, Bits “R” Universal
22. Practice writing Functions (lab)
23. Complexity, Universality Principle
24. Work on Turtle Game HW (lab)
25. Midterm Review and Recap
26. Midterm Exam – In class 1 hr
27. Lab Canceled
28. Recursion – Examples/Uses
29. Practice Recursion in Proc'g (lab)
30. Reasoning: Algorithm Correctness
31. Artificial Intelligence – Jeopardy
32. Pair Programming Intro (lab)
33. Internet Structure + WWW
34. Pair Programming Project (lab)
35. DNS and Large Scale Systems
36. Presidents Day
37. Guest Lec: Ladner, Accessibility
38. Searching, Page Rank, Big Data
39. DIY Instruction in HTML (lab)
40. Debugging Principles
41. Guest Lec: Popovic Games 4 Good
42. DIY Instruction in CSS (lab)
43. Data, Meta-data, XML
44. Try Out XML Structuring (lab)
45. Database Structure
46. Crypto + Steganography
47. XML & XSL practice (lab)
48. Review and Recap
49. Help w/Final Project (lab)
50. Party Hats & Pizza

d) **Evidence of Student Work** Creativity is the #1 “Big Idea” and it was #1 in the class. As soon as the rudiments of programming in Processing were presented, the students were assigned the task of inventing two interactive “art works.” (See the class web site, linked below.) Later in the course students were asked to invent and implement a game, in order to practice the basics of working in pairs. Though they were not using game development software, Processing is effective enough that they could focus on the concepts, even if totally glitzy features were not available.

An example was Matar Gokiburi developed by two women, one a native Spanish speaker (for whom matar is “kill”), and one a native Japanese speaker (for whom gokiburi are “cockroaches”). Analogous to a typing drill, the player must type the letter fast as the cockroaches scurry around the screen.



#### e) **What Worked And Didn't**

*The Language.* UW's CS Principles was an outlier among the pilots in that it did not use a drag and drop programming language. The Processing decision was motivated by preferring a “real” tool used in practice. Whether that is advantageous can be debated. (Students who later took CSE142 *Computer Programming in Java I* said it “was no problem ... we'd seen most of it in *CS Principles*.”) But the main criticism against symbolic programming– the so called “syntax barrier” – did not present any significant difficulties. The Processing IDE is very helpful, “Did you forget a semicolon?”, and students quickly got past those issues. Indeed, there was wide agreement that Processing is just plain fun.

*Lightbot.* Using the video game Lightbot 2.0 as a first-day exercise to introduce basic computational ideas – specify-then-execute, sequential specification/sequential execution, limited instruction repertoire, functions, repetition, etc. – was enormously effective, and smoothed the way into Processing well.

*HTML/CSS.* Because of the sequencing glitches mentioned below, teaching HTML and CSS wasn't fitting into the schedule, so students were given a cheat sheet and a 3-slide introduction, and asked to learn it on their own. This turned out to be highly successful. Students did learn it on their own. But, they were in effect learning the point that they could teach themselves technology. That turned out to be an unintended, but very valuable result.

*Abstraction.* It's the #2 idea in the "Big Ideas" list, and it was a topic addressed in some way most weeks. Though I didn't test closely for their understanding, my perception based on the discussions suggests a good level of understanding by the 2/3 point of the class.

*Sequencing.* The rational order for the topics was revised after a few weeks in order to prepare for the debut of the Watson system and the *Jeopardy* competition. Though it was sensible to do this, it seemed like the course lost momentum for a week or two after that.

*Concept Gradient.* Using Lightbot and the supportive world of Processing, it was possible to introduce concepts without going too fast, but that didn't prevent me from assigning a couple of "killer" homework exercises. Most students survived them, but they need to be fixed. Curiously, these mistakes didn't ruin most students' self-confidence.

## f) Links, Resources, Acknowledgments

The pilot was co-taught with Susan Evans, a Seattle high school teacher; she gave a few lectures, designed projects and vetted all assignments. The teaching assistant was Brandon Blakeley, a graduate student in CSE.

The class Web Site: [www.cs.washington.edu/cse120](http://www.cs.washington.edu/cse120)

Blog written while teaching CS Principles: [csprinciples.cs.washington.edu/blog/](http://csprinciples.cs.washington.edu/blog/)

Class documents zipped together:

[www.cs.washington.edu/education/courses/cse120/11wi/dist/distribute.html](http://www.cs.washington.edu/education/courses/cse120/11wi/dist/distribute.html)

The Processing Home Page – notice the Exhibition and Learning Links

[www.processing.org](http://www.processing.org)

Lightbot 2.0 – it's accessible from many sites

[armorgames.com/play/6061/light-bot-20](http://armorgames.com/play/6061/light-bot-20)

## Analysis

---

The five pilots were taught “under a microscope.” Extensive surveying, weekly conference calls, instructor summaries and other methods were used to gather data to understand how well the courses were succeeding. The final report on the pilot 1 courses [6] gives complete data. For the purposes on this article, a few selected graphs seem most useful.

### Attracting Diverse Students

As indicated in other articles in this issue, a major goal of the CS Principles effort was to attract a population of students that includes many who are not predisposed to study computing. The field not only battles negative stereotypes, but its labor pool must be enlarged both to meet expected demand, but also to introduce more diverse opinions, especially considering the importance of social media.

The results are very encouraging.

### Gender and Racial/Ethnicity Distribution

Most of the pilots were obligated to advertise for students because the CS Principles course was new. These efforts were generally directed at under-represented populations.

The gender balance over the five pilots showed 59% women, 41% men among the 654 students reporting [6].

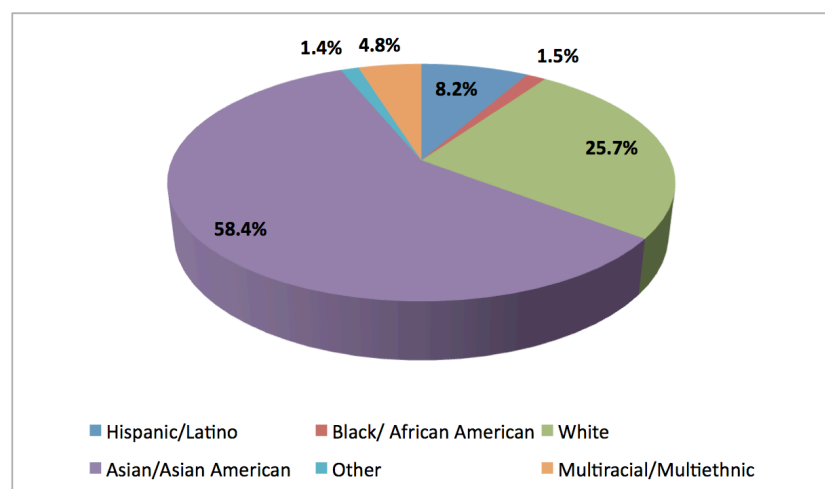
Racial and ethnic diversity is summarized in the following chart. Notice that because of the characteristics of the computing field, the Asian/Asian American group is not considered under-represented. Overall the under-represented population was 11.1% of the total of 661 students reporting [6].

### Overall Student Success

A careful analysis of student performance over the five pilots summarized the diversity issue as follows:

*In addition, the results of a Mann-Whitney test indicated that statistically significant differences*

*were found between the final grades of non-underrepresented students (median=3.30) and underrepresented students (median=3.00). Non-underrepresented students had an average grade of a “B+” compared to underrepresented students who had an average*



grade of “B.” However, the effect size was below the accepted standards for even a “small” effect size indicating a significant difference but not a substantial difference between the final grades of non-underrepresented and underrepresented students.

Lastly, there were no significant differences between the final median grade for males and females; this indicates that, based on final grades as a measure, females learned as well in the course as males. [6]

### Apportioning The Seven Big Ideas

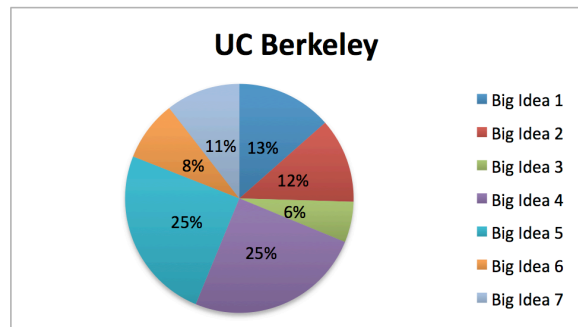
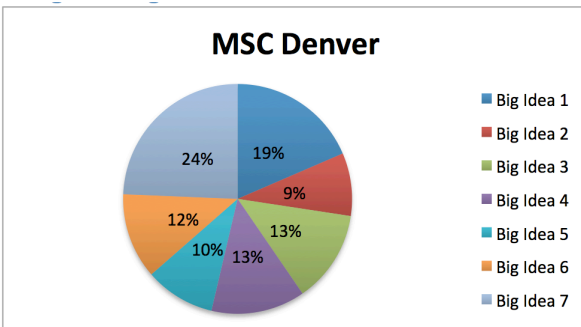
Though the Seven Big Ideas may have been the intellectual foundation of the pilots, there was no specification of how much class time should be allocated to each. It may have been a goal of the piloters to balance the topics across the course, but many other considerations affected the course content. In the end, there was variation among the five, although collectively a rough balance was achieved.

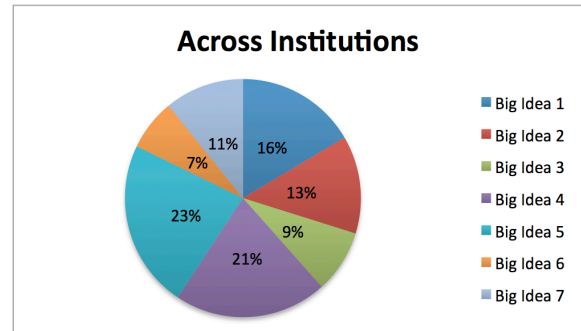
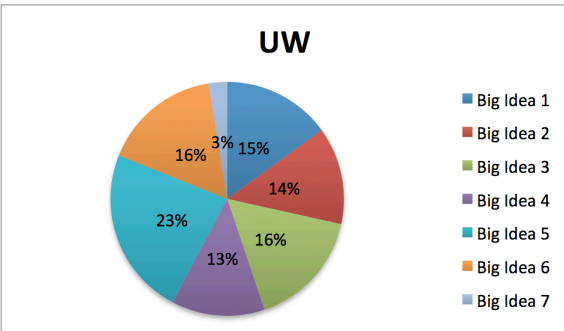
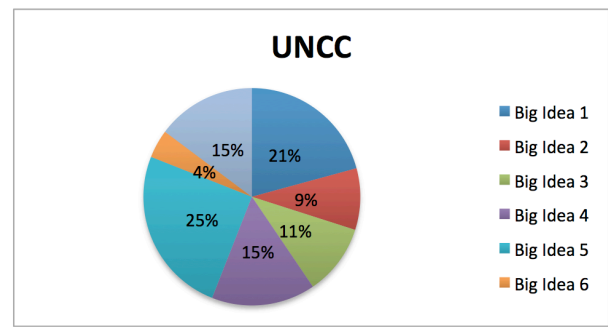
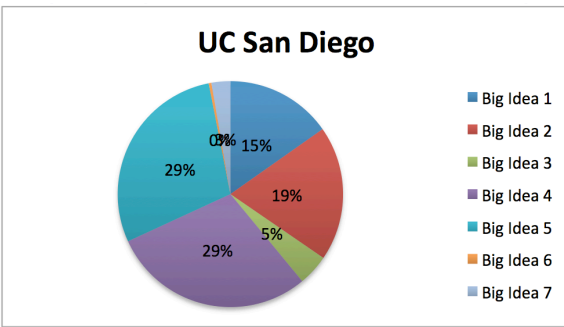
The following five figures show the allocation of class time to the Seven Big Ideas during the term by institution. The sixth figure is the summary over all pilots. The data was reported at the end of each week by the instructor, and aggregated by Kathy Haynie [6].

As outlined above, the Big Ideas in use during the 2010-11 pilots of CS Principles related to 1. Creativity, 2. Abstraction, 3. Data and Information, 4. Algorithms, 5. Programming, 6. Systems, 7. Impacts of computing.

It is perhaps not surprising that taken together the pilots allocated the most time to Idea 5, programming. It’s a difficult topic to learn. But, CS Principles is not a programming class, and programming accounted for less than a quarter of the contact time, narrowly edging out Idea 4, algorithms. The least amount of the class concerned Big Idea 6, “systems.” This may indicate the instructor’s view of where technology falls into the curriculum, but there was also some confusion on the part of some piloters (and the Advisory Board) about which topics were included under the term “systems”. The Advisory Board later clarified Idea 6 as follows: 'Internet: The Internet pervades modern computing', as well as other minor wording revisions.

Overall the report [6] indicates that the pilots were successful at “packaging” the ideas and practices into a course, and within that, engaging under-represented groups.





## Acknowledgments

The work described herein was funded by the National Science Foundation grant [0938336](#). Additionally, early development of UC Berkeley's pilot was funded by a grant from Lockheed Martin. All funding is gratefully acknowledged.

## References

- [1] Seven Big Ideas of Computer Science, 2010, [www.csprinciples.org/home/about-the-project](http://www.csprinciples.org/home/about-the-project)
- [2] Six Computational Thinking Practices, 2010, [www.csprinciples.org/home/about-the-project](http://www.csprinciples.org/home/about-the-project)
- [3] CS Curriculum 2008 Update, ACM, [www.acm.org/education/curricula-recommendations](http://www.acm.org/education/curricula-recommendations)
- [4] Simon, B. Cutts, Q. *Peer Instruction: A Teaching Method to Foster Deep Understanding*. Communications of the ACM, Feb 1, 2012.
- [5] Cutts, Q., Esper, S., Simon, B. *Computing as the 4th "R": A General Education Approach to Computing Education*. In proceedings of the International Computing Education Research Conference, 2011.

[6] Kathleen Haynie and Vonda Johnson, “AP CS: Principles Course, Evaluation Report for Year 2”, July 12, 2011.

### **Author information**

Lawrence Snyder  
Department of Computer Science and Engineering  
University of Washington  
Seattle WA 98195-2350  
snyder@cs.washington.edu

Tiffany Barnes  
Department of Computer Science  
University of North Carolina at Charlotte  
Charlotte, NC 28223  
tiffany.barnes@uncc.edu

Dan Garcia  
Department of Electrical Engineering and Computer Sciences  
Computer Science Division  
University of California, Berkeley  
Berkeley, CA 94720-1776  
ddgarcia@cs.berkeley.edu

Jody Paul  
Department of Mathematical and Computer Sciences  
Metropolitan State College of Denver  
Denver, Colorado 80204  
jody@acm.org

Beth Simon  
Department of Computer Science and Engineering  
University of California, San Diego  
La Jolla, CA 92093-0404  
bsimon@cs.ucsd.edu

**Categories and Subject Descriptors:** K.3.2 [Computers and Education]: Computer and Information Science Education – Computer science education, Curriculum

**General Terms:** Experimentation, Human Factors, Design, Management, Measurement

**Keywords:** Computer science education, pedagogy, CS Principles