Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

<table>
<thead>
<tr>
<th>Applicant</th>
<th>Scott Portnoff</th>
</tr>
</thead>
</table>
| Institution     | Downtown Magnets HS  
1081 West Temple Street, Los Angeles, CA 90012 |
| Email           | srp4379@lausd.net; srport@alum.mit.edu |
| SIGSCE (ACM)    | # 5511971 |
| Project Title   | Teaching HS CS as if the Rest of the World Existed |
| Brief Project Description | Design, Implementation and Rationale for a HS pre-APCS Curriculum of Interdisciplinary Central-Problem-Based (ICPB) Units that Model Real-World Applications. Units address the complexities of solving central problems in the fields of Astronomy, Molecular Modeling, Political Science (Suffrage/Elections), Environmental Science, Bioinformatics/Evolution, Music, and Holocaust Studies. |

Contents

Introduction, 2

Part I. Problems with Secondary CS Courses:
  Gender Imbalances and Low Enrollments, 5

Part II. Rationale and Guidelines for an Introductory Pre-APCS Programming Course that Structures Units around Central Problems, 8

Part III. Topics for Central-Problem-based Interdisciplinary Units, 10

Part IV. Standards for a CS Curriculum, 12

Part V. Concluding Thoughts:
  The Importance of Context in a Pre-APCS Curriculum, 20
Introduction

Two major problems have plagued Computer Science education at both the secondary and post-secondary levels for at least the last two decades: (1) low enrollment numbers and (2) a severe gender imbalance. Educational commentators and researchers since 1990 have speculated that curriculum issues might be a major factor contributing to gender imbalances. They argued for an educational paradigm that included real-world contexts, with inclusion of social and humanistic values. Although statistical evidence suggests that gender imbalances at the high school level may simply be an artifact of very low enrollment numbers overall, this does not invalidate the remedy these educators proposed; in fact, their proposed solution may be an essential piece for boosting enrollment numbers for all secondary students in CS.

CS as an academic field also has a public relations problem: the popular perception is that its major applications are limited to gaming, social networking, hand-held devices and the Internet. Worse, this limited view of the scope of CS is reinforced by CS educators and decision-makers themselves. Even though CS has wide application across the academic, commercial and social spectrums, very little, if any, of this content makes its way into curricula.

Since mathematics is the academic field closest to CS, it may be instructive to look at its curricular models, specifically IMP (Interactive Mathematics Project). In the 1990s, NSF funded the creation of this alternative HS math curriculum, a 4-year program of problem-based mathematics that replaces the traditional Algebra I / Geometry / Algebra II / Trigonometry / Pre-calculus sequence. Each IMP unit has a specific mathematical focus and is structured around a complex central problem into which other topics are brought in as needed for its solution. Because the prominence of CS stems from its facility as an applied science, curricula easily could lend themselves to model significant problems in many cross-curricular settings. In contrast to IMP, whose extensions of central problems to real-world situations are often a stretch, examples for modeling problems outside of CS are abundant.

Units either taught or planned include the following:

1. **Around the World in 24 Days** (BYOB), in which students create a simulation to visualize the phenomenon described at the end of Jules Verne's book: that a traveler circumnavigating the globe traveling east will experience 1 day more than those remaining at the starting point.

2. **Galileo's Revolution** (Alice), in which students create both Copernican and Ptolemeic models of the Solar System and observe whether they account for (a) the phases of Venus and (b) Mars' retrograde motion.

3. **The Right to Vote** (BYOB), in which students write a program to tally sample results of the 2000 presidential election using Palm Beach's flawed butterfly ballot.

---

1 The curricular materials can be accessed at the URL below:

**ECS-BYOB Moodle Course:**

Click on the *Login as a guest button.*

Enrollment Key: portnoffBYOB
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

4. The DNA Double Helix (BYOB), in which students learn the mathematics behind computer graphics ops (translation, rotation, scaling, mirroring) in order to write a 2-D molecular modeling application for the 4 DNA bases and position them to visualize the hydrogen bonding between the opposing anti-parallel strands of the double helix.

5. Bioinformatics/Evolution (Excel): Construction of Phylo-genetic trees from genomic and protein databases, modeled after SDSC Biology Workbench. Other candidates: DNA Fragment Assembly (used in the Human Genome Project); SNP (Single Nucleotide Polymorphism) Mapping of Genes; Population Genetics

6. Music (Processing): Music Visualization Software, such as Synthesia or Music Animation Machine.


8. History/Holocaust Studies: IBM's active support for Nazi Germany's extermination programs through its punch card and Hollerith machine technologies.

Each central problem was also placed within a social context. In the solar system unit, students studied Bertolt Brecht's play Life of Galileo, whose major conflict is scientific truth vs. authority/convenience. The Right to Vote included not only study of the 2000 election, but of the early 20th-century women's suffrage movement. The back-story for the DNA Double Helix unit was the history of the complex interactions between Watson / Crick and Rosalind Franklin, whose x-ray diffraction data was crucial to construction of their DNA model, but to whom they never gave credit. The historical and literary content is not simply tacked on: as an integral part of each unit, it places each problem within a social / historical context, and helps explain why solving the central problem is important in the first place.

Consideration must be given as to whether such units will cover the requisite CS standards. Were there recognized standards, or standardized pre-APCS courses, one might be able to respond. Although CSTA has proposed standards, there are three problems with its objectives and outlines. First, it is a top-down approach, which – lacking any research to support it – consists simply of educated guesses. Second, the standards are topics without contexts; it is inadequate to stipulate Iteration, without specifying which of dozens of types of tasks to which this concept can be applied. Third, algorithms are dealt with on only the most rudimentary level. The author would argue that a curriculum such as the one described in this proposal, given a broad enough range of fields and problems, might be a source for deciding what might be the specific concepts and skills we want to teach at the high school level. Moreover, a curriculum composed of central problems explored in depth lends itself to unforeseen, but fascinating, sub-problems that crop up on the way to crafting the larger solutions.

Programming Standards can be divided into at least 4 levels:

3. Abstraction: Object-Oriented Model.
4. Algorithms: Object-Oriented Problem-Solving Strategies

The sub-problems encountered so far in the ICPB units are more often on the levels of Abstraction and Algorithms, and turn out to be the more powerful and engaging ideas in each
unit. Studying these sub-problems and ideas can give rise to specific standards CS educators may never have previously considered.

The largest obstacle to implementing a course of ICPB units in an introductory course is students' lack of a programming foundation; without such, the programming concepts in each unit will remain opaque. For many pedagogical reasons, students must gain competence and skill in applying programming concepts starting from Day 1. This year, the author has structured the freshman introductory course as one semester of programming in Processing, to be followed by a semester of ICPB units. The expectation is that student retention into the 2nd-year course – which will continue the presentation of ICB units – will increase.

The materials for the first four units should be in a completed state by July 2012, and the first drafts of subsequent units will be finished by that date as well.

Over the next three years, the project will be evaluated vis-à-vis pedagogy, student learning, and student retention into subsequent CS courses.

By July 2013, application will be made to the University of California to approve the curriculum as satisfying the A-G requirements as a Mathematics or Interdisciplinary Elective.
Teaching HS Computer Science as if the Rest of the World Existed  
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Part I. Problems with Secondary CS Courses: Gender Imbalances and Low Enrollments

In a research study of differences between male and female CS students at CMU, Margolis et al. found:

While most of the male students describe an early and persistent magnetic attraction between themselves and computers, women much more frequently link their computer science interest to a larger societal framework. Nearly half of the women we’ve interviewed attach their interest in computer science to other arenas, such as medicine, education, space exploration and the arts...

Curriculum helps set the tone and, unfortunately, most computer science programs in their early years are narrowly focused on programming and the more technical aspects of the field, with applications and multidisciplinary projects deferred to the very end. This gives beginning students the false message that computer science is "only programming, programming, programming," abstracted away from real world contexts. Feminist educator Sue Rosser [1990], from her investigation of gender and science education, argued that "insuring science and technology are considered in their social context ... may be the most important change that can be made in science teaching for all people, both male and female" (p.72.) Computer science professor Dianne Martin [1992], in her article In Search of Gender Free Paradigms for Computer Science Education, discusses "a premise for the gender bias in computer science: the existing educational paradigm that separates studies of science, math, and computer science from studies of the humanities, starting in the secondary schools." She speculates that an integrated approach to computer science would attract more women students, and that "greater attention [should be paid] to values, human issues, and social impact as well as to the mathematical and theoretical foundations of computer science." 1

If Martin were correct about the causative effect of the compartmentalization of academic fields, then one would expect gender disparities to show up in STEM areas across the board. However, this is not the case: the gender gaps at the HS level have narrowed or closed for most STEM fields. Moreover, data for the STEM fields from the 2010 College Board AP Exams show a positive correlation between higher enrollment numbers for all students and gender balance.

<table>
<thead>
<tr>
<th>AP EXAM (2010)</th>
<th>N</th>
<th>% Female</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics C E&amp;M</td>
<td>14191</td>
<td>23.1%</td>
</tr>
<tr>
<td><strong>Computer Science A</strong></td>
<td><strong>20120</strong></td>
<td><strong>19.0%</strong></td>
</tr>
<tr>
<td>Physics C Mechanics</td>
<td>31973</td>
<td>26.2%</td>
</tr>
<tr>
<td>Economics – Micro</td>
<td>51601</td>
<td>42.9%</td>
</tr>
<tr>
<td>Physics B</td>
<td>67312</td>
<td>35.0%</td>
</tr>
<tr>
<td>Calculus BC</td>
<td>78998</td>
<td>40.5%</td>
</tr>
<tr>
<td>Economics – Macro</td>
<td>83146</td>
<td>45.0%</td>
</tr>
<tr>
<td>Environmental Science</td>
<td>86650</td>
<td>55.4%</td>
</tr>
<tr>
<td>Chemistry</td>
<td>115077</td>
<td>46.7%</td>
</tr>
<tr>
<td>Statistics</td>
<td>129899</td>
<td>50.4%</td>
</tr>
<tr>
<td>Biology</td>
<td>172512</td>
<td>57.8%</td>
</tr>
<tr>
<td>Psychology</td>
<td>177708</td>
<td>57.3%</td>
</tr>
<tr>
<td>Calculus AB</td>
<td>245867</td>
<td>48.4%</td>
</tr>
</tbody>
</table>
These data suggest that the severe gender gap in Computer Science education at the high school level may simply be an **artifact of low enrollment numbers overall**. This conclusion is bolstered by research showing that, on a country-by-country basis, "improved social conditions for women were related to improved math performance by girls"². That is, educational gender imbalances appear to have a systemic societal cause. It is therefore unlikely that a solution to gender imbalance problems in CS education will have anything to do with gender directly.

This conclusion does not, however, invalidate Margolis, Rosser and Martin’s speculative remedies, namely that: (a) **CS be taught from the start using "real world contexts"**; (b) **the sciences be "considered in their social context"**; and (c) **attention be paid to "values, human issues and social context"**. Rather than alleviate gender imbalance problems directly, these strategies might actually be what are required to increase absolute enrollment numbers.

Because CS is an elective, the simplest explanation for low enrollments is that students see little value to CS in the real world, a viewpoint that is hardly limited to young people. Were educational professionals, officials on state boards of education, and society at large cognizant of the application of CS to a broad array of fields – in contrast to the popular perception that the scope of major applications of CS is limited to the fields of gaming, social networking, hand-held devices, and the Internet – there would be no question that CS would be taught as a required **academic** subject at the secondary level.
Most other AP STEM subjects, such as Calculus AB, Statistics and Macro Economics, have healthy HS enrollments and either a small or no gender gap. What could possibly make these subjects sexy, and CS not, is a mystery; however, what is clear is that CS has a P.R. problem. A glaring deficiency in CS education is curricula which do little to seriously connect their content to other academic and social areas. Although many CS educators have adopted such innovative and engaging student-learning IDE's as Alice and Scratch/BYOB, HS curricula using these tools nevertheless still consist of problems that are short and make-believe. The emphasis is on students learning programming control structures and concepts, but devoid of serious connection or application to the real world. When students finish such a course, they may have gained some programming skills, but they have little idea how they can apply this knowledge outside the classroom. And so a severe problem in terms of recruitment into the field becomes an even more dire one in terms of retention of students for more advanced study.

One philosophical point regarding content: when we speak of "Computer Science" at the high school level, what we are really talking about is Programming and the considerations that go into writing programs. Bearing this out, the College Board's framework for the APCS-A exam – the only standardized computer science course nationwide – is a Programming course. With few exceptions, APCS-A is not concerned with the "science" aspect of CS; indeed, it is only after having acquired programming skills that other aspects of CS will make sense. Therefore the introductory CS curriculum that I am proposing in the following pages is one that centers about programming.

As a CS teacher, it is not my goal that the students in my classes go on to become CS college majors. Computer Science – like math, biology, chemistry, history and English – should be seen as an integral part of a balanced high school liberal arts education. My working hypothesis is that a rigorous CS curriculum that connects CS to a full spectrum of topics that students already perceive as important – having studied them in their other classes – will positively affect the enrollment numbers.
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of
Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Part II. Rationale and Guidelines for an Introductory Pre-APCS Programming Course
that Structures Units around Central Problems

An alternative to a traditional curriculum sequenced according to programming control structures and concepts is one that consists of carefully sequenced units that revolve around central problems which students solve over a period of several weeks using whatever CS tools are needed.

In mathematics, IMP (Interactive Mathematics Program), created with the support of the NSF in the 1990s, is one curriculum created around this paradigm. IMP is a 4-year program of problem-based mathematics that replaces the traditional Algebra I-Geometry-Algebra II/Trigonometry-Precalculus sequence.

**IMP** units are generally structured around a complex central problem. Although each unit has a specific mathematical focus, other topics are brought in as needed to solve the central problem, rather than narrowly restricting the mathematical content. Ideas that are developed in one unit are usually revisited and deepened in one or more later units.

For example, although the real-world application is a stretch, the complex problem posited by **IMP’s High Dive** unit (Year 3) has students calculate at what point a circus performer on a turning Ferris wheel should dive so as to land in a tub of water on a moving cart.

The **HIGH DIVE** problem. Students extend right-triangle trigonometric functions to the circular functions, learn about the graphs of the sine and cosine functions, study polar coordinates, inverse trigonometric functions, and the Pythagorean identity, and study the physics of falling objects.
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Building a similarly structured CS curriculum will use the following guidelines:

1. Computer Science’s natural relationship to other disciplines is known as Software Engineering, a field which uses programming and CS concepts, along with expert knowledge of specific target topics, to model and solve societal problems and needs. The topic areas for which software engineering methods might be applied may themselves be within the realm of CS, however the vast majority are not.

2. Each unit lasts several weeks and revolves around solving a central problem in the target topic area. A software solution evolves in a scaffolded way, utilizing whatever CS concepts and control structures may be required. Various strategies are attempted at each step to solve problems, and their advantages and disadvantages are analyzed.

3. A drill-like series of short practice problems (using a self-motivating model, similar to Codingbat.com) are retained from the traditional educational paradigm to serve as necessary supplemental exercises – like problem sets at the end of math lessons – to help students cement CS concepts and skills whenever they are introduced or used in new ways.

4. One way instructors can foster connections is by having students create small scale versions of engaging and complex real world applications. With little imagination, students should be able to envision logical extensions of their projects to the already existing and more complex programs from which student exercises were inspired.

5. In order to solve a unit’s central problem, students may need to review, or be introduced to, new concepts in geometry, trigonometry, biology, physics and so forth. This reflects typical considerations that software engineers encounter in their day-to-day work, i.e. programmers must not only be proficient in their own field, but must have knowledge of the specific (non-CS) systems they are modeling in order to write accurate, robust and logically organized programs. The pedagogic advantage for students in needing more than token exposure to other academic fields to solve a central problem is that the multiple contexts allow students more handles to recall and integrate what they learn.

6. Units also place the central problem within a social or historical context. This helps students to not only understand how to solve the central problem, but provides them an explanation why solving the central problem is important in the first place. These so-called back-stories may utilize Literature, Theatre, History, Social Studies, Economics, Film and Art in making those connections. This is NOT done in a tokenistic way. This part of the unit may take up to several days and in some cases more than a week of class time. Students are assessed on this material in a serious way, such as writing an essay in response to a choice of several prompts. A major consideration should be that instructors make connections to topics students learn in other academic courses.
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of
Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Part III. Topics for Central-Problem-based Interdisciplinary Units

Units were developed beginning in Spring 2011 using BYOB, Alice, Excel or Processing. The following list include units either taught or in the planning stages. A Moodle course containing materials for units already taught can be accessed using the URL found in the Footnote on Page 2:

1. **Around the World in 24 Days (Geography).**
   *Relativity of Time Perception when Circumnavigating the Globe*
   Students create a simulation to visualize the phenomenon described in Jules Verne's book, that a traveler circumnavigating the globe traveling east, will experience 1 day more than one remaining at the starting point. Likewise, a west-bound traveler will experience 1 day fewer.
   Excerpts from Jules Verne's *Around the World in 80 Days.*

2. **Joshua at Giv’on, Commanding the Sun to Stand Still: Galileo's Revolution (Astronomy).**
   *Modeling the Copernican and Ptolemeic Planetary Systems to illustrate (a) the phases of Venus and (b) retrograde planetary motion.* The Inquisition and Galileo's Recanting of the Copernican Model.
   Discussion of Bertolt Brecht's play *Life of Galileo.*

3. **The Right to Vote (History).**
   *Optical Scan Technology and Voting Machines.* Democracy in the context of both the women's suffrage movement and the contested 2000 Florida Presidential election (Bush vs. Gore).
   Discussion of the films *Recount* and *Iron-Jawed Angels.*

4. **The Discovery of the DNA Double Helix (Biology / Structural Bioinformatics / History of Science)**
   *Computer Graphics and Molecular Modeling Software as applied to the Hydrogen Bonding of bases between the two anti-parallel strands of a DNA molecule.* This structural model is used to visualize and explain various aspects of point mutations. History of the complex interactions/clashes between the researchers involved in the discovery of the structure of DNA: Rosalind Franklin, Maurice Wilkins, Frances Crick and James Watson.
   Discussion of the BBC film *Life Story / Double Helix.*

5. **IBM's Strategic Contribution to the Efficiency of Nazi Germany's Final Solution (History, Ethics, Holocaust Studies)**
   *Before the invention of computers, PUNCH CARD TECHNOLOGY was used to solve database-related problems.* The sorting and tabulating algorithms used with these cards were direct precursors of methods used for present-day databases. IBM and its German subsidiary were active participants in the processing of population data used to identify, transport and exterminate the Disabled, Jews, Gypsies, Homosexuals and Communists throughout Europe from 1933 through 1945.
   Excerpts from the book: *IBM and the Holocaust.*
   Film/Book: *Diary of Anne Frank.*
   Film/Book: *Sarah's Key* (deportation of a young Parisian girl and her family)
   Film: *A Film Unfinished* (footage from the Warsaw Ghetto)
   Film/Play: *Copenhagen* (conflict between physicists Niels Bohr and Werner Heisenberg)
6. Evolution and Social Reaction (Evolution, Genetics, Bioinformatics).
Use of Genomic Databases and Software Tools to Align DNA and protein sequences from related species and build Phylogenetic (Evolutionary) Trees. Study and modeling of Bioinformatics algorithms (LCS, Global Alignment, Local Alignment, Scoring Matrices, Clustering) used by UCSD's Biology Workbench. Discussion of the play *Inherit the Wind*.
Part 1 of the Unit has been developed and is available at the Bioinformatics Activity Bank. [http://teachingbioinformatics.fandm.edu/node/75](http://teachingbioinformatics.fandm.edu/node/75)

7. The pure and simple truth is rarely pure and never simple (Environmental Science)
• Predator-Prey Population Simulation Software, and the unintended consequences of human activity, such as overfishing.
• Earthquake Epicenter Algorithms using Triangulation.

8. On the Road (Geometry / Geography)
GPS and Routing Programs
• A GPS program based upon triangulation of satellite data, equations for calculating longitude and latitude on a sphere, and a geographic database.
• A routing program using Dijkstra's shortest path algorithm (like MapQuest).

9. Music Visualization (Music Animation Machine and Synthesia)
Synchronization of the orchestral instruments of musical works – such as Beethoven's 7th Symphony or Vivaldi's Four Seasons (Winter) – with colored geometric shapes moving across the screen that represent pitch and duration.
Discussion of Film: *In Search of Beethoven* (documentary 2009)

**Other Candidate Unit Topics:**
1. DNA Fragment Assembly (used in the Human Genome Project)
2. SNP (Single Nucleotide Polymorphism) Mapping of Genes
Part IV. Standards for a CS Curriculum

Two questions that can be asked of an Interdisciplinary Central-Problem-Based (ICPB) curriculum are:

What is the curriculum’s effectiveness?
Are all necessary programming standards being hit?

We can answer the first question by assessing students' abilities to use specific skills and concepts to solve variations of problems in new settings. The second, however, cannot be answered because a list of comprehensive Computer Science standards, with accompanying research-based pedagogical teaching strategies, does not yet exist. This can be summed up in three problems currently faced by HS CS Education:

1. There is little research on which to judge the effectiveness of pedagogical strategies.
2. There are no accepted or recognized national or state standards; and
3. There is no credible certification process, nor credential training programs, for Computer Science instructors.

These shortcomings are interdependent. If there is no research that tells us what concepts we should be teaching at each grade level, any standards we develop can only be speculative. And if there are no reliable standards, there can be no agreement on what should be taught and when. Therefore reliable pedagogic content for credential programs cannot be developed. In addition, there can be no standardized assessments, either handwritten or computer-based.

The problem of grade-level standards may in fact not be a serious obstacle. In mathematics, the content of the traditional sequence-based HS curriculum can be taught using the IMP paradigm with at least equal efficacy. On the other hand, CSTA's 2006-7 K-12 Curriculum Model has a reasonable list of objectives-topics for Programming Languages (Level II) and Program Design and Problem Solving (Level III), but specifics and details are lacking, pedagogic examples are simplistic or shallow, and immediate contexts (multi-varied programming uses) as well as broader compelling real-world contexts are completely missing. What is really needed is a comprehensive document addressing the complexity and depth for each topic and grade-level on a par with NCTM's Principles and Standards for School Mathematics (The Mathematics subject area, particularly the subdisciplines/strands involving proofs, is probably the best pedagogic model to inform HS CS educators because of the similarity in thinking and problem-solving that successful students need to master.)

What might be a good strategy for identifying specific CS Programming standards? Previous attempts at compilations of standards seem to have been top-down, similar to the strategy of Backwards Planning in which teachers first write an assessment and then devise course content and activities. What this approach cannot provide, though, is the relative importance of each topic since these can only be educated guesses based on one's experience and expertise.

This proposal will instead employ a bottom-up approach using an ICPB curriculum from which programming standards will then be extracted. Given a large enough sampling of units covering a broad range of subject areas, one could then distill those concepts that are most widely and often used. The importance of CS is due almost exclusively to its bent as an applied science in a sweeping range of areas. It may turn out that a curriculum composed of central problems explored in depth might be a source for deciding what might actually be the specific topics and skills we want to teach at the high school level.
What should standards look like? As an example, consider the modulus operator, which surprisingly is mentioned in none of the three levels of CSTA's *Model K-12 Curriculum*. In contrast, three of the four units that I developed and taught in Spring 2011 used a circular buffer, and two of those explicitly implemented the iterative wrap around using the modulus operator (the other was implicitly embedded in the IDE). A standard for the modulus operator might be stated as follows:

Given a positive integer modulus divisor, students will be able to list the range of possible remainder values. Students will know how to use the modulus operator to restrict the possible values of a property variable for a circular buffer, such as one used for the implementation of a clock, calendar, or circle.

The process of building a computer program can be viewed from at least four somewhat overlapping levels\(^4\), where decisions made at one level affect the implementation at others:

1. **Coding** consists of the nuts and bolts implementation of Programming Language Concepts, line-by-line. These include Variables and Data Types, Loops, Iteration, Conditionals, Boolean Expressions, Scope, Recursion, Classes/Objects, Methods/Parameters, and the like.
2. **Structure** concerns itself with Organization, Modularization and Hierarchy of Methods.
3. **Abstraction** involves creating an Object-oriented software Model of the problem that is accurate, apt, consistent, insightful, extendible and predictive.
4. **Algorithms** comprise theoretical and economical ways to implement a solution consistent with the Object Model.

In an ICPB curriculum, students would encounter most or all Programming Language standards in every unit. Standards in the beginning units of a course would be presented in their most basic form, and complexity scaffolded as solutions are built. Further along in the course, there will be critical points in the piecing together of solutions where standards are used in new and unexpected ways; these become the focal points for student growth at these later stages.

Categories 2-4 can be grouped under the larger heading of Program Design. Although program design standards are more difficult to articulate, they manifest in larger and lengthier projects, where one is required to critically evaluate and decide amongst competing, sometimes contradictory, design ideas. Although the tendency in introductory programming courses is to emphasize mechanics at the Coding level, it is really at the levels of Abstraction/Modeling and Algorithms where the most powerful and engaging ideas emerge, affording students an overview of the big picture. A first course in HS programming should give students glimpses of meaningful insights into all of these levels. Below are three examples depicting the kind of design standards that can arise as one works through a long project.

**Example 1** is from a unit that tackles the problem of how to write the class method `dialNumber(startN, endN, direction)` for an Alice ComboLock object. Although this unit is from a curriculum no longer used, its solutions reveal some important principles that arise while solving long central problems. The unit can be found at:
The lesson (in Unit 4) entitled *Example ComboLock, Part II: dialNumber* discusses the different cases one needs to consider to devise an algorithm that does the calculations algebraically. The algorithm is straightforward, but not trivial:

<table>
<thead>
<tr>
<th>EndingN &gt; StartingN</th>
<th>StartingN &gt; EndingN</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Difference = EndingN - StartingN</td>
<td>1. Difference = StartingN - EndingN</td>
</tr>
<tr>
<td>2. If dialing left, nLoops = Difference</td>
<td>2. If dialing left, nLoops = 40 - Difference.</td>
</tr>
<tr>
<td>3. If dialing right, nLoops = 40 - Difference.</td>
<td>3. If dialing right, nLoops = Difference</td>
</tr>
<tr>
<td>4. Call turnOneNumber nLoops times</td>
<td>4. Call turnOneNumber nLoops times</td>
</tr>
</tbody>
</table>

This is followed by a lesson entitled *Example ComboLock, Part IV-A: a better dialNumber method; Modifying turnOneNumber* which discusses the shortcomings in the algebraic solution and offers an alternative:

Although the *dialNumber* method works, it does have a major disadvantage: you can only dial to a new number *if you know the current position of the dial*. In other words, every time you call the method, you have to also pass a parameter for the last number you dialed. If the argument for that parameter is not correct, the method will dial the wrong number!

When you dial a number on a *real* combination lock, you don't look at the current number and then try to calculate how many positions you need to turn the dial: you simply dial to the final number. You can do this because the lock knows which number the dial is positioned at. For example, if the dial is on the 10, the lock knows this - if it didn't, the lock wouldn't work.

In computer science, unlike mathematics, you often have the option of crafting solutions for your software object (the *model*) that reflect how a real object (the *system*) actually works. ... We can make the *comboLock* behave like a real lock by creating a *new property variable* called *currentNumber* to store (keep track of) the position of the dial every time the dial moves.

Implementing a property variable that keeps track of the dial's current position means we can dispense with the *startingN* parameter and greatly simplify the *dialNumber* method, whose body will now consist of a loop with just one instruction: keep turning in the specified direction until *currentNumber == endingN*. Of course, this requires that supporting methods also implement the property variable, as well as one other change: the dial needs to be implemented as a circular buffer in order to deal with the boundary condition where incrementing 39 results in 0 and decrementing 0 results in 39.\(^\text{15}\)

This kind of lesson deals with Abstraction and Algorithm standards and their interdependence, and contrasts them with a purely algebraic solution. The contrast between the two implementation strategies is intended to emphasize the power of Abstraction to simplify a program's implementation and make it more intuitive. A first, general attempt at characterizing this type of standard might be:
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Students will implement suitable property or state variables for modeled objects in order to give them an intelligence or self-awareness that can be subsequently used in intuitive, non-algebraic methods for governing their behavior.

Example 2 is from one of the current BYOB ICPB units: Around the World in 24 Days, which simulates 3 people’s experience of day/night cycles over a 24-day period. One circumnavigates the globe traveling east, one travels west, and one remains in the same spot. The non-traveler sees 24 days, the east traveler sees 25 days, and the west traveler sees 23 days.

The lesson starts with a stationary object on a non-rotating earth and a moving sunrise. In setting up our measurement system, we used the standard unit circle convention: 0° is due east and angular position increases as one travels east (CCW). We also used the mod function to normalize angle positions so that they remain within the range 0 <= angle < 360 as the sun crosses the 0° due East boundary point. Our animation uses 96 photo images (taken every 15 minutes over a single 24-hour period), so the distance between each sunrise position is 3.75°.

One of the first problems to solve is how the program will detect an object encountering a sunrise. One way to detect such an event is to determine when the object lies between the two edges of a sector made up of the last and current positions of the sunrise. The conditional expression for this is:

\[
\text{Pos}_{SR} < \text{Pos}_{obj} \quad \text{AND} \quad \text{Pos}_{obj} \leq \text{Pos}_{lastSR}
\]

Because the sun travels west, in decreasing angular direction, the normal state of affairs is that \(\text{Pos}_{SR} < \text{Pos}_{lastSR}\). However, when the sector straddles the boundary point, \(\text{Pos}_{SR} < 360\) and \(0 \leq \text{Pos}_{lastSR}\) because the angle values are normalized. Therefore \(\text{Pos}_{SR} > \text{Pos}_{lastSR}\), and the conditional expression above will fail; therefore we need to consider a second conditional expression for this boundary case.

There are (at least) two ways to think about how to write this new expression. A first is to imagine the sector bounded by values that have not been normalized. The AND expression below on the left describes the sector if the angle position of the east side >= 360. The AND expression on the right describes the sector if the angle position of the west side < 0:

\[
(\text{Pos}_{SR} < \text{Pos}_{obj} \quad \text{AND} \quad \text{Pos}_{obj} \leq \text{Pos}_{lastSR} + 360) \quad \text{OR} \quad (\text{Pos}_{SR} - 360 < \text{Pos}_{obj} \quad \text{AND} \quad \text{Pos}_{obj} \leq \text{Pos}_{lastSR})
\]

An alternative is to use the normalized values, but imagine the sector divided into two parts separated by a line at 0°. In this case, the west part of the sector is described by the AND expression on the left, and the east part is described by the AND expression on the right.

\[
(\text{Pos}_{SR} < \text{Pos}_{obj} \quad \text{AND} \quad \text{Pos}_{obj} < 360) \quad \text{OR} \quad (0 \leq \text{Pos}_{obj} \quad \text{AND} \quad \text{Pos}_{obj} \leq \text{Pos}_{lastSR})
\]
In either case, the expressions can be simplified because, by the time the program encounters these expressions, the angles will have been normalized and there is no need to check whether Pos_{lastSR} >= 360 or Pos_{SR} < 0. One can therefore cross out the portions indicated below:

\[
\begin{align*}
(P_{SR} < P_{obj} \text{ AND } P_{obj} <= P_{lastSR} + 360) \text{ OR } (P_{SR} - 360 < P_{obj} \text{ AND } P_{obj} <= P_{lastSR}) \\
(P_{SR} < P_{obj} \text{ AND } P_{obj} < 360) \text{ OR } (0 <= P_{obj} \text{ AND } P_{obj} <= P_{lastSR})
\end{align*}
\]

Below is the final code for detecting the sunrise in our stationary earth model:

```java
boolean detectSR_StationaryEarth() {
    if (PosSR < Pos_{lastSR})
        return PosSR < Pos_{obj} AND Pos_{obj} <= Pos_{lastSR}
    else
        return PosSR < Pos_{obj} OR Pos_{obj} <= Pos_{lastSR}
}
```

Notice that the only difference between the two expressions is the substitution of OR for AND.

One way to express this situation as a standard would fall under the heading of parsimony and simplification of Boolean expressions. But this only captures one aspect of what is going on. What is engaging about this example is the apparent contradiction found in the similarity of the two expressions and the apparently simpler, less rigorous OR expression used for the more complex boundary point situation. However, the simplification was only possible because of the implicit normalization system set up earlier; without that, the expression would have remained more complex. The algorithm standard therefore lies in articulating the spillover effect of the normalization system into simplifying algorithms further down the line, i.e. the dependency of algorithms on earlier implemented design details.

**Example 3.** The unit then leads students into constructing a similar model for a rotating earth, but taking into account that the sunrise remains in a fixed position, while the object’s angular position changes as it is swept along with the earth’s rotating surface. In this case, we detect a sunrise by determining when the *sunrise* lies between the two edges of a sector made up of the last and current positions of the object. In the rotating earth simulation, the earth’s angular direction is the opposite of the sunrise’s direction in the stationary earth model. Therefore, the normal state of affairs is that Pos_{lastObj} < Pos_{currObj}. The algorithm for a rotating earth is:

```java
boolean detectSR_RotatingEarth() {
    if (Pos_{lastObj} < Pos_{currObj})
        return Pos_{lastObj} < Pos_{SR} AND Pos_{SR} <= Pos_{currObj}
    else
        return Pos_{lastObj} < Pos_{SR} OR Pos_{SR} <= Pos_{currObj}
}
```
It turns out that each algorithm works well for its respective model for objects as long as they aren't moving.

When we introduce traveling objects, the algorithm continues to work for the (seemingly more complex) rotating earth simulation because nothing fundamental has changed: the sunrise position remains static, and objects move, whether they are traveling along the earth's surface or not.

However, for the (seemingly simpler) stationary earth model, at some point during its circumnavigation, an object fails to detect a sunrise when traveling east, and adds a double sunrise event when traveling west. The breakdown happens because our algorithm was not designed to compare the positions of two moving quantities: the sunrise and (now non-static) traveling objects.

When an object is traveling east, it is moving in the opposite direction from the sunrise. At some point, its exact location (the small red dot) lies on the western side of the sector. At the next polling event, the sunrise would normally encompass the object if it had remained still. However, the object has moved just beyond the sector's eastern side, causing the program to miss the sunrise event. Note that this phenomenon is independent of the speeds of either the sunrise or the traveling object.

One way to reach a solution lies in reimagining the stationary earth model so that either the sunrise moves or the objects move, but not both. If we hold the sunrise constant, the new model will be analogous to the rotating earth model. We can make the sunrise appear to remain stationary mathematically by subtracting its value from all quantities used in the calculations. We can then discard the detectSR_StationaryEarth() method and replace it with a modified version of the detectSR_RotatingEarth () method that will handle both cases:

```java
boolean detectSR () {
    local pos_lastObj = Pos_lastObj - Pos_lastSR
    local pos_currObj = Pos_currObj - Pos_SR
    local pos_SR = Pos_SR - Pos_SR (i.e. 0)
    if (pos_lastObj < pos_currObj)
        return pos_lastObj < pos_SR AND pos_SR <= pos_currObj
    else
        return pos_lastObj < pos_SR OR pos_SR <= pos_currObj
}
```

Note that one subtracts the value of the last SR – not current SR – from the last position of the object. Mathematically, we are not simply subtracting the same value from all three quantities.
Rather we are expressing the values for each of these three points so that they will be mathematically relative to the reference frame of a static sunrise. Since the position captured in the $\text{Pos}_{\text{lastObj}}$ variable was taken relative to the sunrise position captured at the same time in the $\text{Pos}_{\text{lastSR}}$ variable, the latter is used as the subtraction quantity.

We can simplify by adding back $\text{Pos}_{\text{SR}}$ to each of the three local variables:

```java
boolean detectSR () {
    local $\text{pos}_{\text{lastObj}} = \text{Pos}_{\text{lastObj}} + (\text{Pos}_{\text{SR}} - \text{Pos}_{\text{lastSR}})$
    local $\text{pos}_{\text{currObj}} = \text{Pos}_{\text{currObj}}$
    local $\text{pos}_{\text{SR}} = \text{Pos}_{\text{SR}}$
    if ($\text{pos}_{\text{lastObj}} < \text{pos}_{\text{currObj}}$)
        return $\text{pos}_{\text{lastObj}} < \text{pos}_{\text{SR}}$ AND $\text{pos}_{\text{SR}} <= \text{pos}_{\text{currObj}}$
    else
        return $\text{pos}_{\text{lastObj}} < \text{pos}_{\text{SR}}$ OR $\text{pos}_{\text{SR}} <= \text{pos}_{\text{currObj}}$
}
```

We can now see that the only quantity that needs adjusting is the edge of the sector marked by the last position of the object-traveler. When the earth is rotating, $(\text{Pos}_{\text{SR}} - \text{Pos}_{\text{lastSR}})$ evaluates to zero, because the sunrise boundary line does not move. When the earth is stationary, $(\text{Pos}_{\text{SR}} - \text{Pos}_{\text{lastSR}})$ is a negative quantity, because the sunrise boundary line travels clockwise (eastward) and $\text{Pos}_{\text{SR}} < \text{Pos}_{\text{lastSR}}$. If the traveler is moving east, the sector will therefore be narrowed slightly. If the traveler is moving west, the sector will be slightly expanded.

The phenomenon itself is not particularly mystifying to students because they can pause their simulations to observe the point at which the original model breaks down. The instructor can present examples of relative motion (e.g. an escalator with two observers, one on and one off the escalator) to help scaffold the reasoning for subtracting $\text{Pos}_{\text{SR}}$ to mathematically change the reference frame so that it is relative to a static sunrise point. However, the best way to convey the model's behavior – both before and after the adjustment – is to step through individual intervals surrounding the breakdown point and observe the values for each sector's two edges and the sunrise boundary point.

Incorporating the reference frame calculations into the solution is a valuable exercise for illustrating the design-test-redesign cycle. However, the most important CS principle that this example brings up is the limitations of discrete models to represent idealized continuous systems or processes.

In a continuous system, our observational reference frame is inconsequential, i.e. it doesn't matter whether we observe the events from the point of view of the traveling object, the sunrise, or some third point: at some specific instant, the sunrise boundary line momentarily encounters the traveler and a sunrise event will be detected.

However, in our discrete model of that system, we used intervals to mathematically calculate an intersection between sunrise boundary and traveling object. This seemed a reasonable
assumption/approximation that replicated correct behavior when the movement of either traveler or sunrise was static. However, when both were moving, it became apparent that our original model was flawed and that the discrete model with identically sized intervals was not an equivalent representation for the continuous system. An adjustment to interval size – based upon inclusion of reference frame considerations and direction of travel – was needed to make the discrete model generate behavior that was then equivalent to the continuous system for all cases we examined (so far) in this project.
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

Part V. Concluding Thoughts: The Importance of Context in a Pre-APCS Curriculum

If we look to the mathematics curriculum as a model, we see that the goal of the traditional vertical sequence is to build a foundation of concepts and skills year by year, laying firm ground for the more advanced strands to follow. Moreover, topics gain in intellectual complexity and interconnectedness as students advance.

CS at the high school level does not comprise a sequence of different strands; rather the single college-level course focuses almost exclusively on the teaching of Programming Language Concepts. Curricular materials written in the last few years for such student programming environments as Alice echo this same focus, but take advantage of a drag-drop interface to avoid syntax errors as a way to reduce the cognitive load. But is the best preparation for a college-level APCS course – i.e. the purpose of a pre-APCS course – simply to teach similar content at a less complex level?

As a college-level STEM course, APCS follows the traditional educational paradigm of extreme compartmentalization. The framework concerns itself with a very limited set of concepts and skills; even connections to CS topics outside of this set are excluded. As such, by the end of this single course, students are unable to write large, complex, expertly organized programs: how could they? They won't know how to do Web programming, either client- or server-side. They won't know how to interface with databases or do file I/O. They will know neither IPC nor TCPIP/socket technology nor multi-threading nor concurrency. They will have no inkling of the kinds of classical algorithms that can be applied to difficult problems. They won't know the basics of GUI design or event-processing, including timers. Because program output is limited to text, they will have no experience with graphic output. Students will have no significant understanding of machine architecture as it pertains to data representation. And they certainly won't know software engineering principles for how to build, maintain and manage large software projects. None of these topics are mentioned in the APCS framework. However, many of them (the ones in bold) are part and parcel of the simplest programs in Processing, BYOB and Alice.

It's a given that the teaching of Programming Language concepts and skills be present in any early CS course – otherwise, students will be unable to learn or create anything useful. However, it cannot be the only consideration. A professional software engineer who has programmed in one language for several years should (ideally) be able to learn and program in a second language without difficulty. This is not due simply to familiarity with Programming Language concepts. It is also due to his/her familiarity with the API: the variety of tasks a program can direct the operating system to perform. Likewise, a major goal of introductory curricula should be that, by the time students take a specialized college level programming class, they will be familiar enough with the kinds of operational tasks programs are capable of to be able to ask questions like:

- How do you get keyboard or mouse input?
- How do you implement a second thread?
- How do you pass a message or generate an event?
- How do you implement a button or a text box?
- How do you catch and process timer events?
- How do you use scope to ensure a variable's integrity?
- How do you do graphic output and display images?
The thrust of the arguments made in this proposal is to give students a broad awareness of the range of fields in which CS has application so that they can put the Programming Language concepts they learn into a larger context. The proposal also advocates placing these cross-curricular problems within a social and historical context so that students will understand the importance of the problems and the contribution CS can make to their solution. Even on a more nuts-and-bolts level, CS education needs to give students a map for how programming concepts fit within a broader programming context vis-à-vis accessing OS tasks. The alternative is the current educational paradigm that caters to the traditional CS demographic, and still leaves most students finishing the APCS-A course with (a) few ideas, if any, for how the knowledge they've learned can be realistically applied; and (b) little perspective on how what they've learned fits into the larger picture of CS programming. Students are therefore left with few external incentives to explore subsequent CS courses.
REFERENCES / ENDNOTES


2 **Women and Math, the Gender Gap Bridged: Social equality frees women to match men** (June 2008). Based on the Research of Luigi Guiso, Ferdinando Monte, Paola Sapienza And Luigi Zingales. [http://insight.kellogg.northwestern.edu/index.php/Kellogg/article/women_and_math_the_gender_gap_bridged](http://insight.kellogg.northwestern.edu/index.php/Kellogg/article/women_and_math_the_gender_gap_bridged)


The exam covers the following topics:
- I. Object-Oriented Program Design
- II. Program Implementation
- III. Program Analysis
- IV. Standard Data Structures
- V. Standard Algorithms
- VI. Computing in Context

4 **Interactive Mathematics Program**.  [http://www.mathimp.org/general_info/intro.html](http://www.mathimp.org/general_info/intro.html)


6 **UCSD, San Diego Supercomputer Center, Biology Workbench**.  [http://workbench.sdsc.edu/](http://workbench.sdsc.edu/)


8 Geology Labs Online: **Virtual Earthquake**  [http://sciencecourseware.org/eec/Earthquake/](http://sciencecourseware.org/eec/Earthquake/)


   SIGCSE'10, March 10-3, 2010, Milwaukee, Wisconsin, USA.

   ©2010 ACM 978-1-60558-183-5/09/03

10 **Cold Spring Harbor Laboratory: DNA Learning Center**  
*The public Human Genome Project: mapping the genome, sequencing, and reassembly. 3D animation*  

11 **The Human Genome Project and SNP Mapping**  

   *An SNP map of the human genome generated by reduced representation shotgun sequencing.*  

12 CSTA has put forth a set of standards ([http://csta.acm.org/Curriculum/sub/Implementation.htm](http://csta.acm.org/Curriculum/sub/Implementation.htm)) in the last several years, but on close inspection, these are more often than not a list of topics. Below are two examples:

   ACM Model **Level II** Curricular Standards

   **Standard 6: The connection between elements of mathematics and computer science, including binary numbers, logic, sets, and functions.**
Teaching HS Computer Science as if the Rest of the World Existed
Design, Implementation and Rationale for a HS Pre-APCS Curriculum of Interdisciplinary Central-Problem-Based Units that Model Real-World Applications

(What specific connections? How will students apply binary numbers, sets, and so on? Why should students know about binary numbers? For representation of data? For bitwise operations?)

ACM Model Level III Curricular Standards
Topic 1: Program Design and Problem Solving.
(Iteration in what contexts; for what tasks? Iteration to solve what kinds of problems?)

In contrast, consider the following two Algebra 1 (California State) standards:

9.0. Students solve a system of two linear equations in two variables algebraically and are able to interpret the answer graphically. Students are able to solve a system of two linear inequalities in two variables and to sketch the solution sets.

19. Students know the quadratic formula and are familiar with its proof by completing the square.

Notice that the math standards are specific concepts and skills detailing how students will demonstrate mastery. The CSTA Model Curriculum Support Documents do have a table of Detailed Descriptions for each topic containing lesson suggestions, but these come nowhere close to describing the specific multi-faceted uses of the programming concepts.

13 National Council of Teachers of Mathematics.
http://www.nctm.org/standards/content.aspx?id=16909

14 Although we always want to be encouraging our students to write their programs in clear and concise ways – and ever modeling such an approach – Clarity, Conciseness and Parsimony are more quality/stylistic issues and therefore applicable to all four programming levels.

15 The MOD function is not used in this solution because the version that Alice does have – IEEE Remainder of a/b – does not give the same result as modulo arithmetic if a is negative; therefore we write our own MOD function in a subsequent unit.