What to Teach? Computational Science as an Improved Model for Science Education

(Submitted 29 March 2008 to Microsoft Research RFP "Computational Education for Scientists – What to Teach?")

Rubin H Landau
Director, Computational Physics for Undergraduates Program (CPUG)
Oregon State University, Corvallis, Oregon 97331
rubin@science.oregonstate.edu, http://www.physics.oregonstate.edu/~rubin

Abstract It is evident that computation has fundamentally changed research and development in most every field of science. This paper argues for concordant changes in science education that extend beyond using computers to teach traditional science better. Results will be presented from surveys of existing computational science degree programs that reveal a consensus of topics essential for the computational education of scientists. Detailed concept maps will be given, showing how educators are combining computer science and applied mathematics with a traditional discipline in order to teach how to solve realistic problems. It is proposed that teaching based on this research-like, problem-solving approach is a more motivating and efficient technique than teaching the various disciplines separately. Because it may be difficult for a single college or university to offer all of the essential topics, a number of early developers are placing eLearning modules from their courses together as part of a national repository.

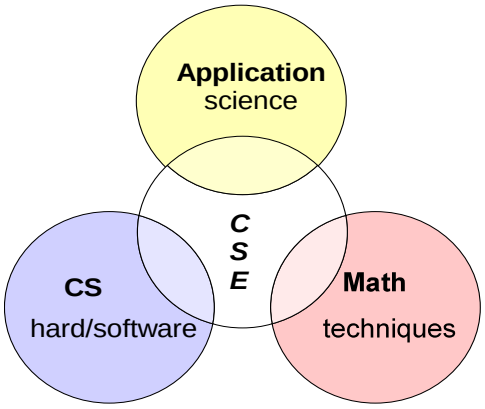

Figure 1. Computational Science & Engineering (CSE) as a multidisciplinary endeavor connecting computer science with mathematics and applications. In this view CSE is more than just the overlap of traditional disciplines.

Changes in Science Drive What and How to Teach

The historically rapid adoption of computation as an essential element in all areas of science and engineering has progressed faster than the concordant changes in education. Although many traditional science disciplines are incorporating computers to enhance education in those disciplines, the education too often presents computation as a "black box" whose inner workings need not be understood. (Some of the published evidence of this comes from Winch' survey's finding that few classes that incorporate computation as a course element actually examine the students about it [Winch].) Given the newfound and fundamental importance of

computation in so many disciplines, and the fact that many graduates of the traditional disciplines actually end up employed as computational scientists (proof to follow), we propose that students and society as a whole would be better served if the traditional science disciplines taught an understanding of computation as part of their science.

In this paper we use the terms "computational science" or "CSE" to denote the multidisciplinary combination of techniques, tools and knowledge developed in the 1970's and 1980's to solve scientific and engineering problems through computer simulation. In terms of disciplines (Figure 1), it is a combination of mathematics and computer science with a field such as physics, chemistry, automotive engineering, *etc*. In our view, there also needs to be a central core (inner circle) based on a common toolset and mindset that draws the disciplines together. Teaching CSE encompasses the integration of modern research tools and subjects into undergraduate education, integration that US News & World Report considers a hallmark of a high quality education. Much of this paper will detail those subjects and tools.

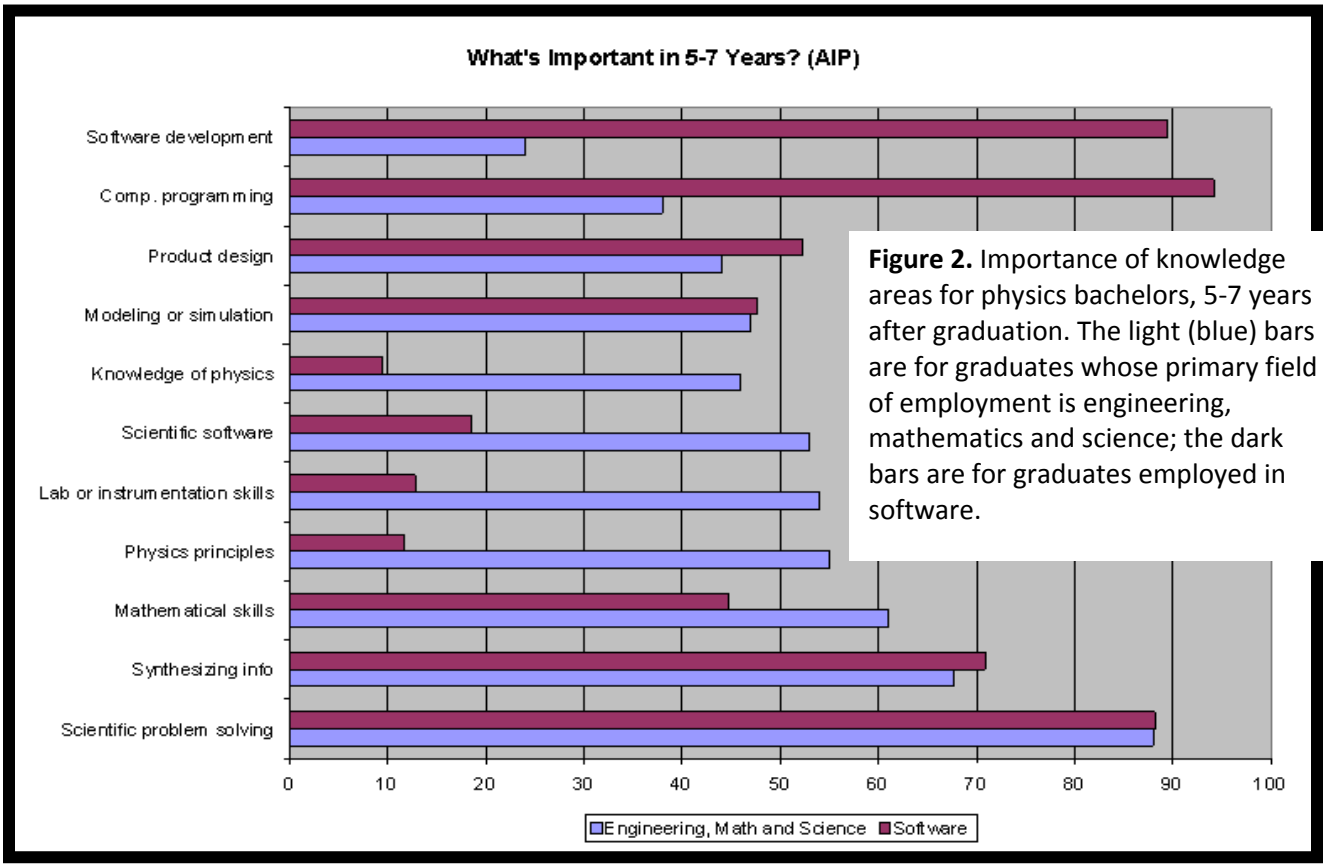

Figure 2. Importance of knowledge areas for physics bachelors, 5-7 years after graduation. The light (blue) bars are for graduates whose primary field of employment is engineering, mathematics and science; the dark bars are for graduates employed in software.

The Need for Computational Education

It is wise to look in the cupboard before making up your shopping list. Likewise, it is prudent to look at what subjects are currently taught and what subjects students need after they graduate, before deciding to change your curriculum. We start by looking at the results of a survey of physics bachelors conducted by the American Institute of Physics that determined which aspects of their education are most valuable in their current employment five years after graduation [AIP]. The results, shown in Figure 2, indicate that for graduates whose

primary field of employment is engineering, mathematics and science, the three most important skills are scientific problem solving, synthesizing information, and mathematical skills. These skills are also highly important for graduates who find employment related to software. While it is to be expected that knowledge of software and programming are most important for graduates in software development, notice how, otherwise, *synthesizing information* is the most important skill for both groups, and that *knowledge of physics is essentially the least important*. (And these numbers come from the AIP!).

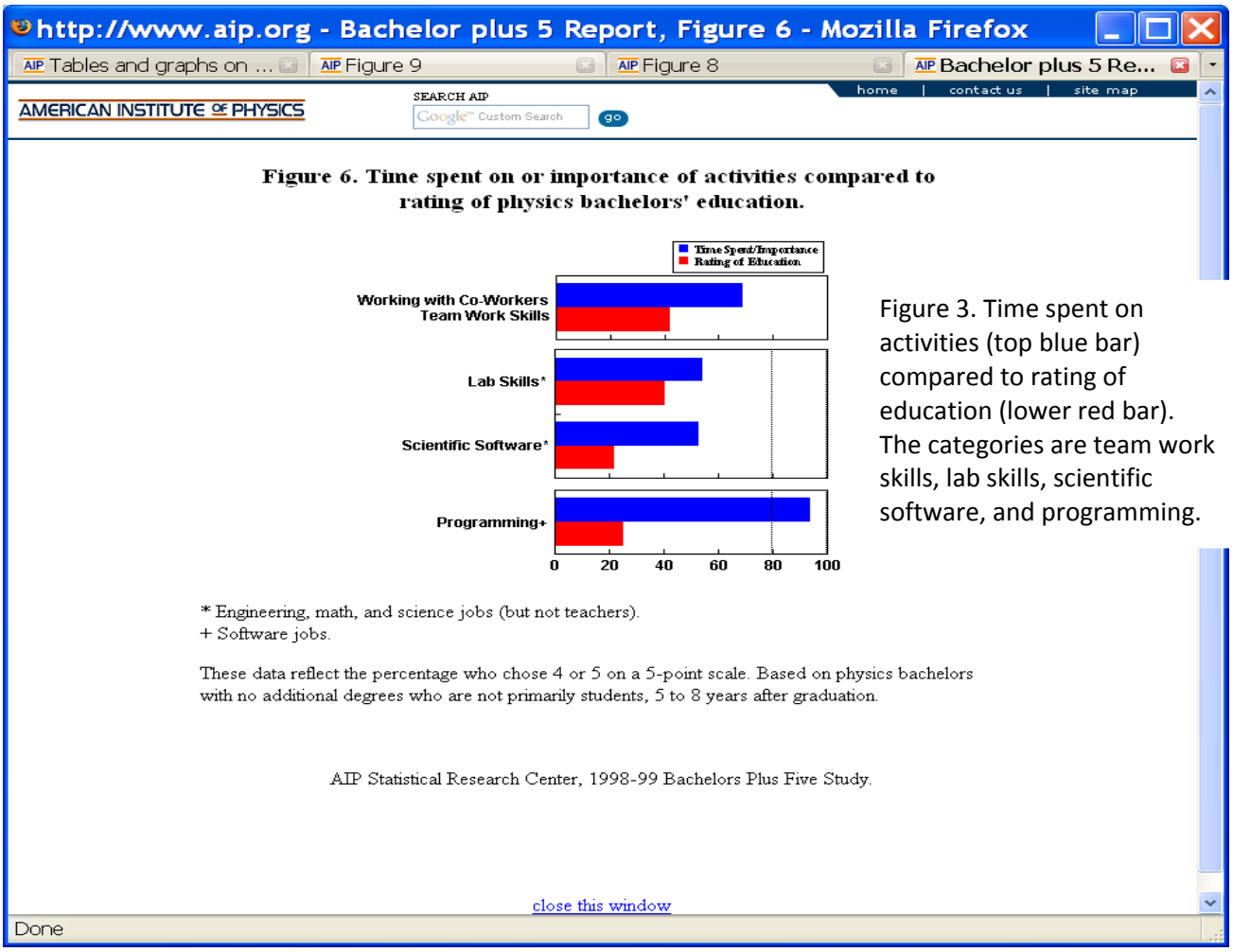

Figure 3. Time spent on activities (top blue bar) compared to rating of education (lower red bar). The categories are team work skills, lab skills, scientific software, and programming.

That same survey also examined (Figure 3) the importance graduates give to various activities as compared to their education in these activities. We see that while physics departments appear to be spending a proper amount of time on lab skills, their student might benefit if there was more time spent on scientific software, programming, and team skills.

Although we have focused on physics here, there is evidence that similar observations hold in other fields. For example, a National Science Board report [NSB] indicates that only 22% of physics and biology undergraduates remain in the field from which they graduate, with the number rising to only 52% at the graduate level. To get an idea of just where they go, look at Figure 4, which shows the fields of employment for the physics bachelors of 2003 and 2004. Notice on the left that most bachelors (53%) are employed by the private sector, with only

12% going into colleges and university. Notice on the right that of those bachelors employed by the private sector, about one third assume non-science, technology, engineering, or mathematics positions, about one third do engineering work, and about one quarter do technology work. This leaves only 13% to do physics or some other science. The conclusion we draw from these figures is that while *a traditional disciplinary may provide good preparation for a student's career, overemphasizing the specific discipline tends to weaken the preparation.*

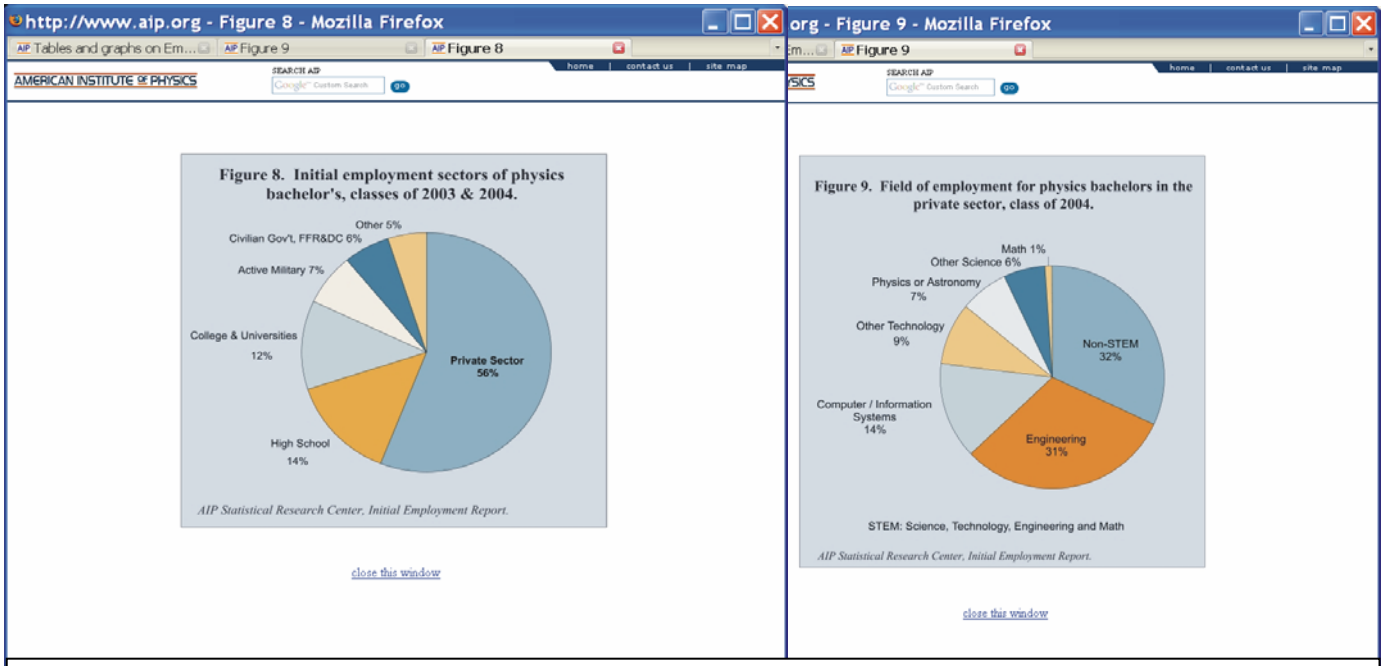

Figure 4. *Left*: the initial employment sectors for the physics bachelors of 2003 and 2004. *Right*: The fields of employment for those graduates employed in the private sector.

Now that we have documented the need for computation in science education, the next step is to determine if that need is being met. Figure 5 shows the results of a survey of the curricula of undergraduate programs in computer science (CS), physics (PH), computational science (CSE), and computational physics (CP) [Y&L]. The survey made the, admittedly crude, categorization of courses as *Computing*, *Mathematics*, *Applications,* and *Other* (whatever is left), and then displayed the results as the average percentage of the total curriculum dedicated to each category. The left column in Figure 5 shows the strong Computing (black) but weak Application (white) components in the Computer Science degree; the right column shows the strong Application but weak Computing components in the Physics degree. The middle columns show that the Computational Physics and Computational Science programs tend to provide a similar, uniform balance among Math, Computing and Application. So while these computational degrees provide less physics than a physics degree and less computing than a CS degree, the balance may be healthier. These numbers agree with our impression (prejudice?) that regular physics undergraduates do not learn enough about computation, and that regular CS undergraduates do not learn enough about math and science (there is of course much math in some CS classes).

In summary, there appears to be a demand for mathematics, computational, and problem-solving skills in the work place, as well as for providing a science education that can be applied to a variety of fields of employment.

Whether individual disciplines, with the natural pressure to get students to understand subjects the way the professors do (and thus teach even more of the discipline), can provide a proper computational education is unclear. However, computational science is still a young and developing field with no set curriculum, and I believe that it will take groups of interested parties to help decide what topics should be taught, and in which courses they may be included.

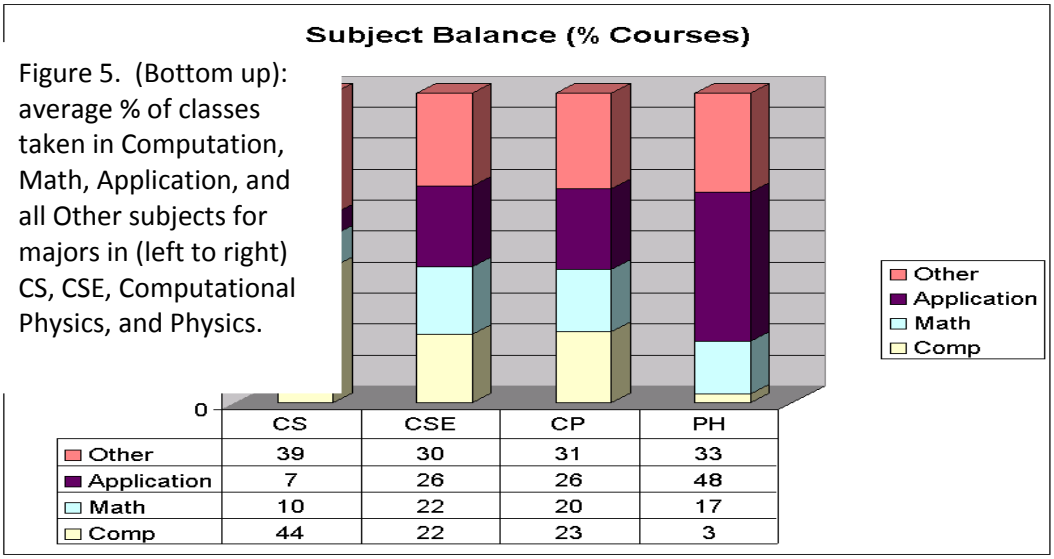

Figure 5. (Bottom up): average % of classes taken in Computation, Math, Application, and all Other subjects for majors in (left to right) CS, CSE, Computational Physics, and Physics.

Present Computational Education Programs

For the past 25 years, individual graduate students have combined courses in mathematics, computer science and a traditional discipline in an ad hoc approach to obtaining a computational education. In that time, computational science has continued to mature, as has the power and pervasiveness of computers in science. This has led to formal degree programs at the graduate level, and more recently at the undergraduate level. Nevertheless, a bachelor's degree in any of the computational sciences is rare, as we see in Table 1 which lists the US and foreign undergraduate degree programs in all the computational sciences.

Table 1. Undergraduate degree programs in all computational sciences.

Computational Math	**Comput Science**	**Computational Phys**	**Foreign Programs**
1. Arizona State			
2. CUNY Brooklyn	1. Stanford (+Math)	1. Houghton C	1. Australian Nat
3. Michigan State	2. SUNY Brockport	2. Illinois State	2. Kanazawawa Japan (CSE)
4. Missouri So State	3. Stevins Inst Tech	3. Oregon State	3. National U Singapore (CSE)
5. Rice	4. UC Berkeley	4. SUNY Buffalo	4. Trinity C, Dublin (CP)
6. Rochester Inst Tech		5. Chris Newport	5. U Calgary (CSE)
7. Seattle Pacific			6. U Erlangen-Nurnberg (CSE)
8. Saginaw Valley State	**Computational Bio**		7. U Waterloo (CSE)
9. San Jose State			8. Utrecht U (CSE)
10. U Chicago	1. Carnegie Mellon		
11. U Illinois Chicago	2. U Pennsylvania		

Yet a degree is not the only way to learn a subject, and so in Table 2 we list computational programs that are not formal degrees, but may well provide an equivalent education.

Table 2. Minors, Concentrations, Tracks, Emphases, Options, Foci, *etc*.

Computational Biology	Computational Science	Computational Physics
	1. Capital	
1. UC Merced	2. Clark	1. Abilene Christian
2. Center CB (Colo)	3. Old Dominion	2. North Carolina State
	4. RPI	3. Penn State Erie
Computational Mathematics	5. Salve Regina	4. U Arkansas
	6. Syracuse	
1. Princeton (App & CM)	7. U Wisconsin Eau Claire	
2. San Diego State (App & CM)	8. U Wisconsin LaCrosse	
3. U Central Florida	9. U Wisconsin Madison	
4. U Nebraska-Lincoln	10. Wittenberg	
	11. Wofford C	

These tables are updated versions of the surveys by Swanson [Swan] and Osman and Landau [Y&L]. These tables may not be complete since we have tried to make them realistic by including only active undergraduate programs, and have excluded some programs that appear to be dual-degree programs, without computational bridge courses that draw the disciplines together. As you can see, the number of programs (51) is small, with Computational Mathematics (which may or may not have a strong Application content) and Computational Science being the most popular ones. While small, this number is about four times larger than the number we found in 2001 when we assembled a similar list, and so the field is growing. Of course, there may well come a time when computation is so integrated into the disciplines that the existence of programs such as these will be viewed as a temporary, transitional trend. Time will tell.

Framework for Teaching Science with Computation

Figure 6 illustrates the scientific problem-solving paradigm that is at the core of computational research. Although diagrams such as Figures 1 and 6 have been shown often enough to become visual clichés, they remain relevant to the focus of this paper since they provide the general structure for computational education. In fact, we believe that the commonality of tools across the computational sciences combined with the common problem-solving mindset is a truly liberating and attractive aspect of computational science because it permits its practitioners to understand and participate in a much wider set of problems than occurs otherwise in the sub specialization of science.

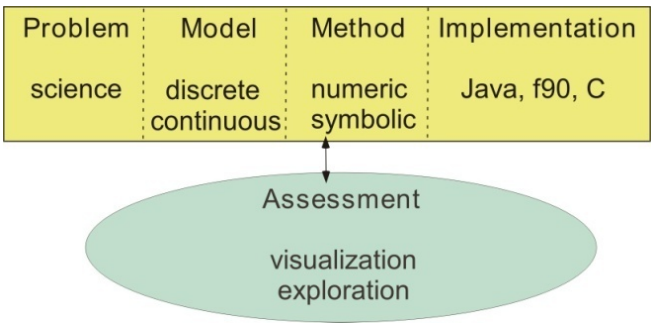

Figure 6. The scientific problem-solving paradigm. A problem is set, the tools from multiple disciplines are employed within context, and the continual assessment aides debugging and steering.

In general, we recommend that computational educational materials be structured around the scientific problem-solving paradigm in Figure 6. This demonstrates where the multiples disciplines are relevant, provides concrete examples that assist in understanding the abstract concepts, and stresses the importance of assessment of the various components through visualization. From a pedagogical perspective, we believe that a Computational X education following the problem-solving paradigm is a more efficient approach to undergraduate education than a pure X education ("X" refers to a specific discipline). Although students may take fewer X classes, they tend to learn the X, CS, and math better when placed in context, and thus get more out of their courses. So even if the number of X courses needs to be reduced to make room for teaching computation, this is compensated for by the increased efficiency of the pedagogy. Furthermore, this approach has been shown to be appealing to a more diverse group than those presently attracted to computer science or physics [LEAD].

A key component of many computational programs is having students get actively engaged with projects as if each were an original scientific investigation, and having projects in a large number of areas. In this way students experience the excitement of their personal research, get familiar with a large number of approaches, acquire confidence in making a complex system work for them, and continually build upon their accomplishments. We have found the project approach to be flexible and to encourage students to take pride in their work and their creativity. It also works well for independent study or distant learning. In order to teach a projects-based course, we employ a combination of lectures and ``over the shoulder'' labs. The students work on and discuss their projects with an instructor, and then write them up as an ``executive summary'' containing sections for

- Problem
- Equations employed
- Algorithm
- Code
- Visualization
- Discussion & Critique

The emphasis is professional, much like reporting to manager in a workplace. Visualizations are important for all the classes, and we teach the use of *Maple/Mathematica, PtPlot, gnuplot, AceGr,* and *OpenDX* for 2-D, 3-D, and animated plots (long lists of resources available to a computation class can be found in [CP-2]). Taken together, this approach produces significant learning, even though we may be ``teaching with our mouths shut'' [Fink].

What to Teach

At present there is no professional organization of computational scientists or accreditation body to decide the proper content for an education that can be called ``computational''. However, a number of interest groups have been working on this, including the NSF-supported Computational Science Curriculum Virtual Institute [CSCVI] and TerraGrid [Terra], the DOE-supported Krell Institute [Krell], the Computing in Science & Engineering

magazine, the Society for Industrial and Applied Mathematics [SIAM] and the American Association of Physics Teachers [CPC].

The CSCVI (in which the author was a member) researched the existing CSE degree programs and confirmed earlier findings [Y&L] of a consensus regarding the basic elements of computational science curriculum. We tabulate the basic elements in Table 3, where "Computational X" refers to courses relevant to a specific discipline (often the central bridge in Figure 1).

Table 3. Common basic elements in existing CSE programs.	
Computer architecture	Computer simulation
Scientific computing systems	Numerical algorithms and analysis
Parallel processing	Combinatorial algorithms
Programming / Parallel programming	Design and analysis of algorithms
Software engineering	Computational X
Scientific visualization	Applied mathematics
Design and implementation of database systems	

The CSCVI project initiated the task of providing a repository of key curriculum subjects using the internet-based Visual Understanding Environment [VUE]. This technology is designed for managing and integrating digital resources in support of learning, and was used to create the concept maps that we will soon show. These maps are graphical representations of subjects or concepts that demonstrate how the concepts relate to one another, and can be used to outline a recommended path to be followed through the materials for mastering them. *The maps below present our major suggestions as to what should be taught when combining science with computation. We recommend that the maps be read at least as carefully as a text.*

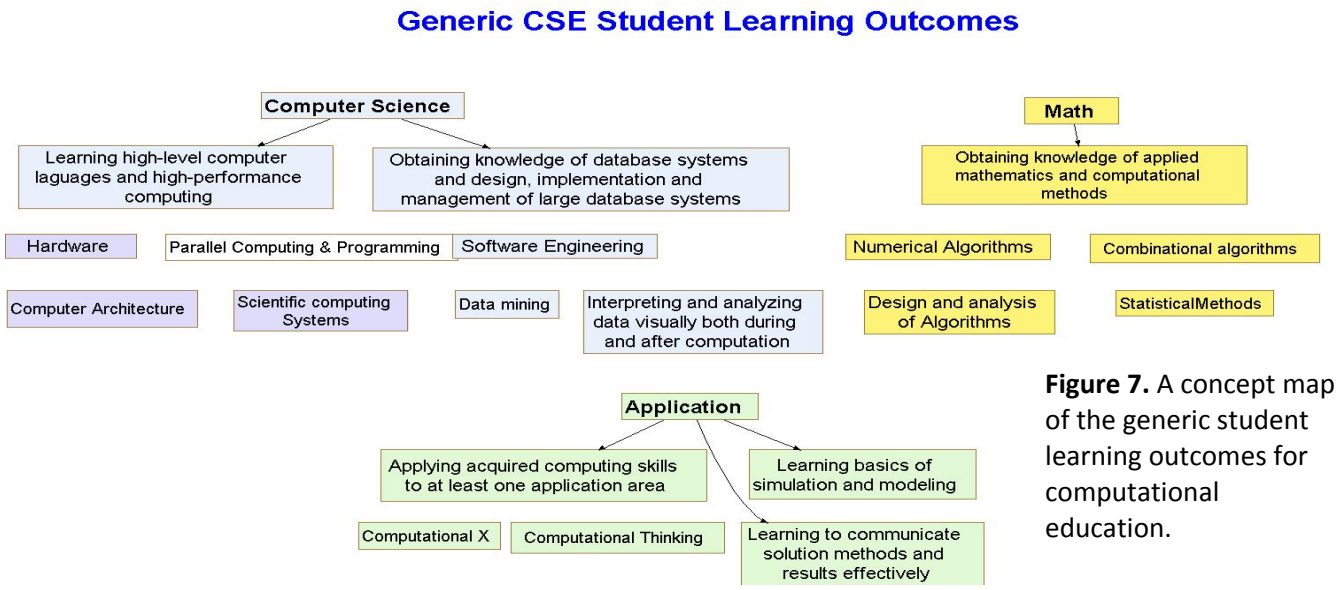

Figure 7. A concept map of the generic student learning outcomes for computational education.

An important prerequisite for the establishment of any course or educational program is the determination of student learning outcomes (SLO's). This is especially true for CSE where it is (all too) easy to expect an individual course to teach students everything they need to know about computation [Yas]. Historically, SLO's have been guided by research needs, that is, in a ``top-down'' fashion in which a graduate student's program committee prescribed what areas need to be studied for an individual thesis project. In Figure 7 we present, in concept map form, some generic SLO's for computation classes and their relations to the basic elements above. As you can see, the SLO's are separated by disciplines, although there is much overlap (as there should be for a multidisciplinary program). Note that in Figure 7 and the maps below we do not try to list all elements in a student's curriculum, but rather just those related to computation. For example, the Math subjects are all applied, while a student would also take some general or pure math classes. Likewise, we do not list traditional subjects for the Application areas.

In Figure 8 we present a generic concept map of the type we would distribute to teachers to help them design a

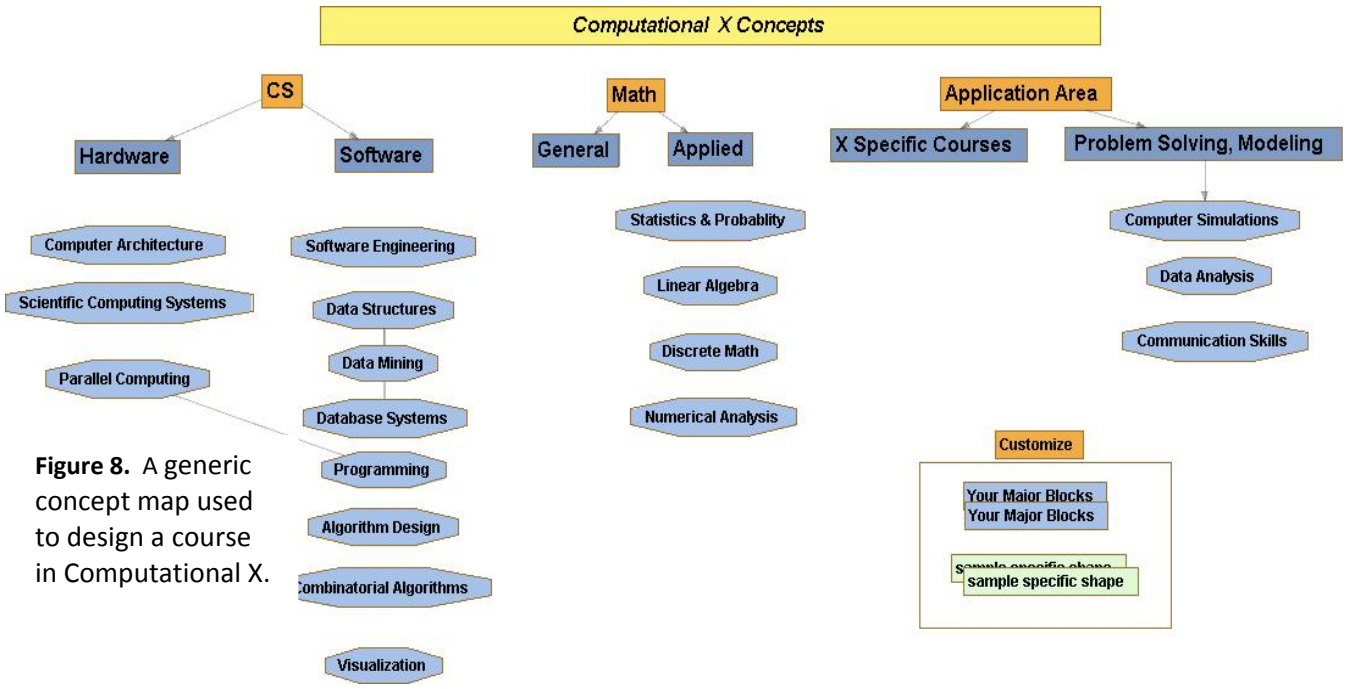

Figure 8. A generic concept map used to design a course in Computational X.

Computational X course. The basic elements of Computational X are suggested, and there are blank blocks included for instructor customization. The VUE software makes it very easy to create and modify maps, as well as to add connecting arrows indicating relations or ordering among the concepts, with further information placed on the arrows if desired. While multiply-connected maps are useful for visualizing a subject, they tend to be overwhelming for a paper such as this and so we have removed most.

In Figure 9 we present a concept map for our *Computational Physics* course and text [Survey]. In Figure 10 we present a concept map for a *Computational Finance* created independently [CSCVI]. These maps are valuable in that they indicate what two computational science teachers are now teaching in two very different courses, and how they are organizing the materials. It is illuminating to contrast these maps. Superficially, we notice that the physics map is laid out in a columnar, top-down style, which makes it easier to read the entries than the radial style used in the finance map; however, it may be easier and clearer to see the relations among concepts in the finance map. We cannot say which is more effective.

When we look at the actual concepts in the two maps, we notice that the Math and Applications elements are very similar in the two, with the finance map giving more application specifics, but fewer CS concepts. Although not shown here, my design of the physics map is to keep the concepts general (which is why they are called "concepts"), but then link individual elements to specific topics. In turn, when the specific topics are linked to the digital content that we have developed, which converts the *concept* map into a *content* map (a research project of the author). The detailed topics are:

Physics 465–6/565–6 Computational Physics (*Computational Physics*, Wiley)

Realistic, Double Pendula*	Quantum Path Integration*
Fourier & Wavelet Analyses	Fluid Dynamics
Predators & Prey: Nonlinear Mappings*	Electrostatic Potentials
Chaotic Pendulum/Scattering*	Parallel Computing (MPI), Heat Flow
Fractals, Aggregation, Trees, Coastlines*	Waves on a String
Bound States via Integral Eqtns	Shock Waves & Solitons
Quantum Scattering, Integral Equations	Molecular Dynamics Simulations
Thermodynamics: The Ising Model	Electronic Wave Packets

Physics 467/567 Advanced Computational Laboratory

Radar Maps of Archaeological Tells	Density Functional Theory
Molecular Dynamics Simulations	Gamow States of Exotic Atoms
Meson-Nuclei p-Space Scattering	Pion Form Factor Data Analysis
Wavepacket-Wavepacket Interactions	Particle Hydrodynamics
Serious Scientic Visualization	Brain Waves Principal Components
Earthquake Analysis	Quantum Chromodyanmaics

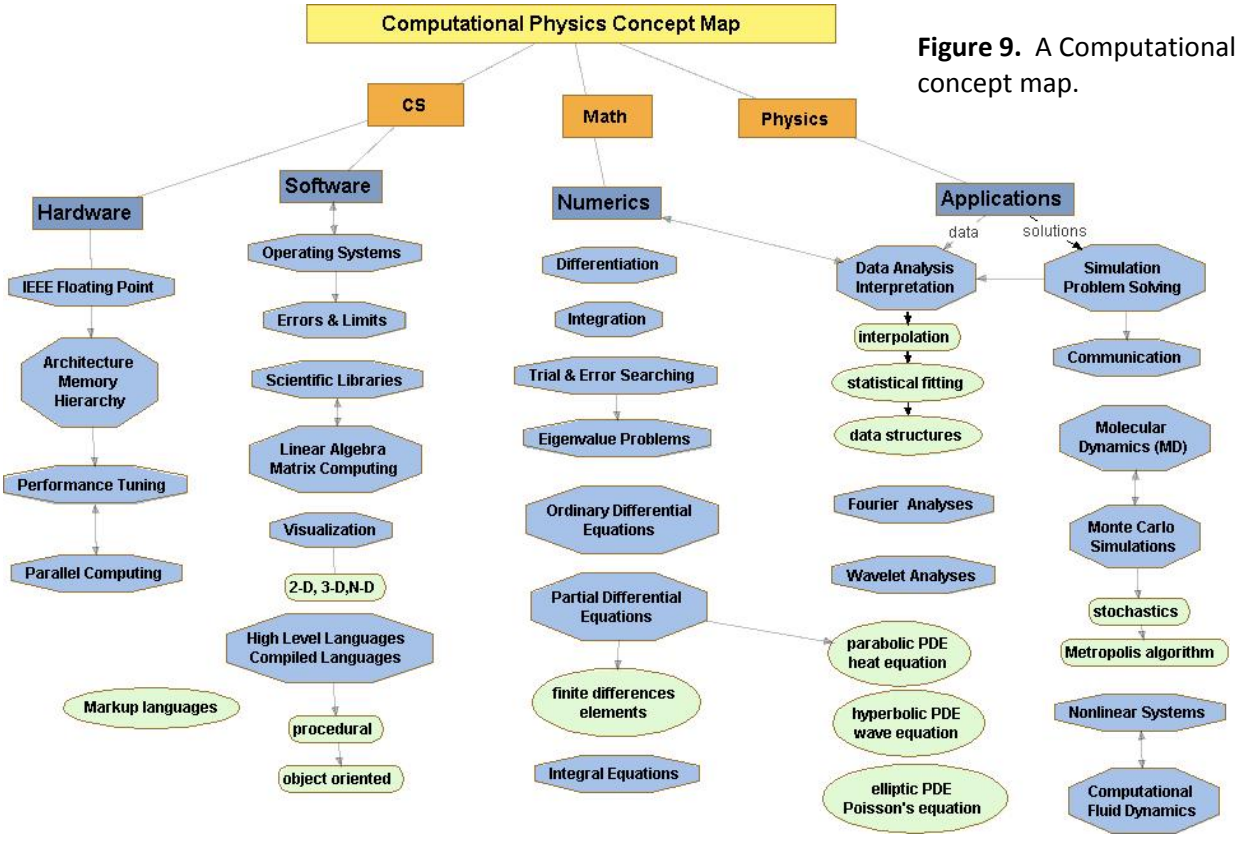

Figure 9. A Computational Physics concept map.

The *Advanced Computational Laboratory* is the equivalent of the classic advanced experimental lab using dusty old graduate theses setups. In the computational version, seniors run dusty deck simulations that were once M.S. and Ph.D. thesis projects. The students get the codes running, investigate some suggested problems, make modifications of the codes themselves, and compare the results to those published in the literature. For many students this is their first experience with truly large and legacy programs, and with reading an article in the scientific literature.

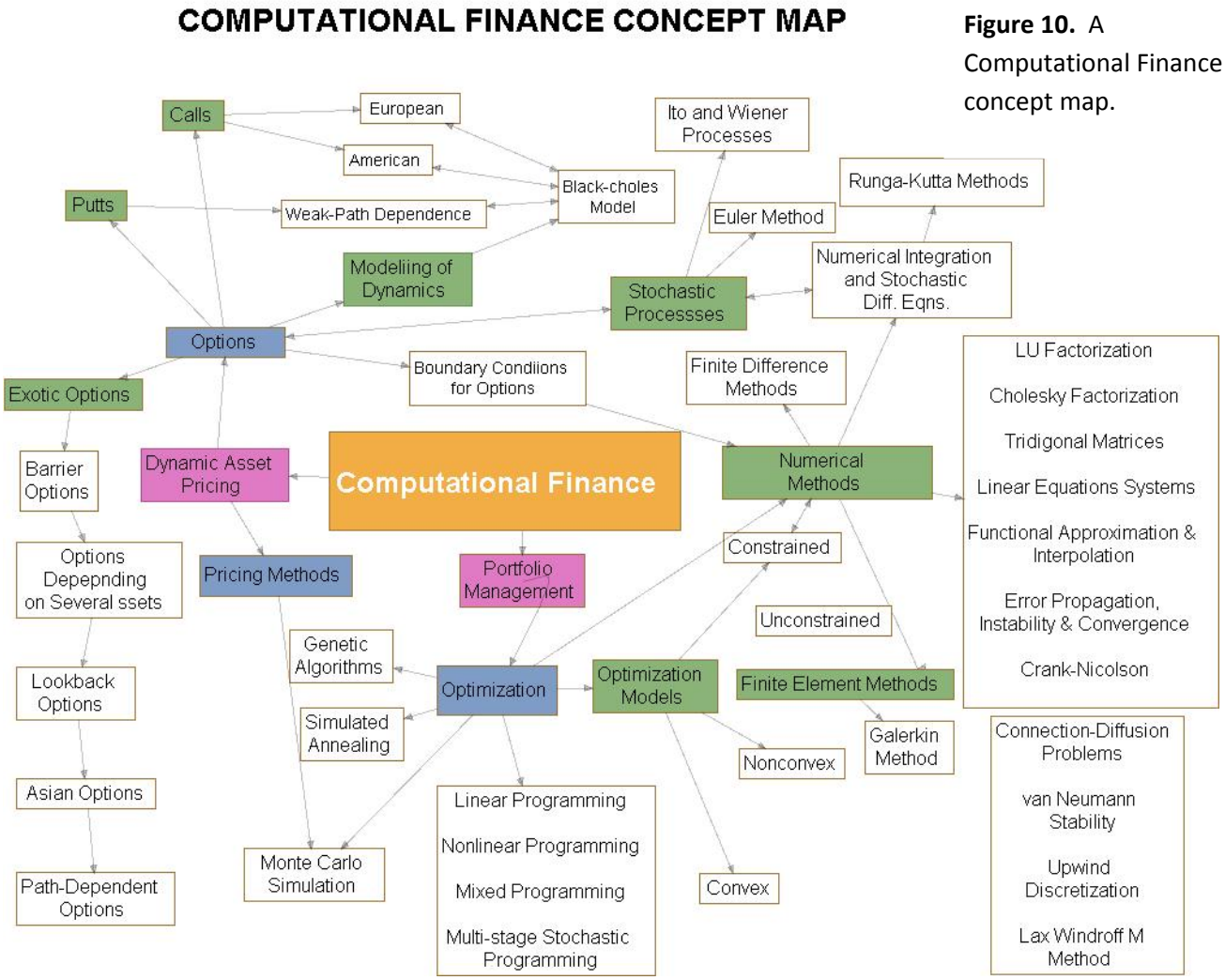

Figure 10. A Computational Finance concept map.

In Figures 11 and 12 we present CSCVI concept maps for individual courses in the *Design of Databases* and in *Numerical and Error Analysis*. Again, one developer chose a radial style and the other a columnar style. The database map is seen to be specialized to just this one particular subject, and thus without the blending of disciplines and with little in common to the other maps show. The numerical analysis class map in Figure 12, while still for just one course, is seen to have a great deal of overlap with the computational X classes, with this map naturally giving more detail than those covering a broader field. Whether all of these numerical analysis topics are best taught in one course or taught in context in Computational X classes, is probably best decided by

local conditions. In my experience, the Computational X classes may teach one or two versions of these methods as needed in some application, but not give a detailed investigation of issues such as convergence and stability. In that case, there would be good reason to teach both classes. As always, one can argue about the best methods and techniques, but these maps are specific examples of what is being taught.

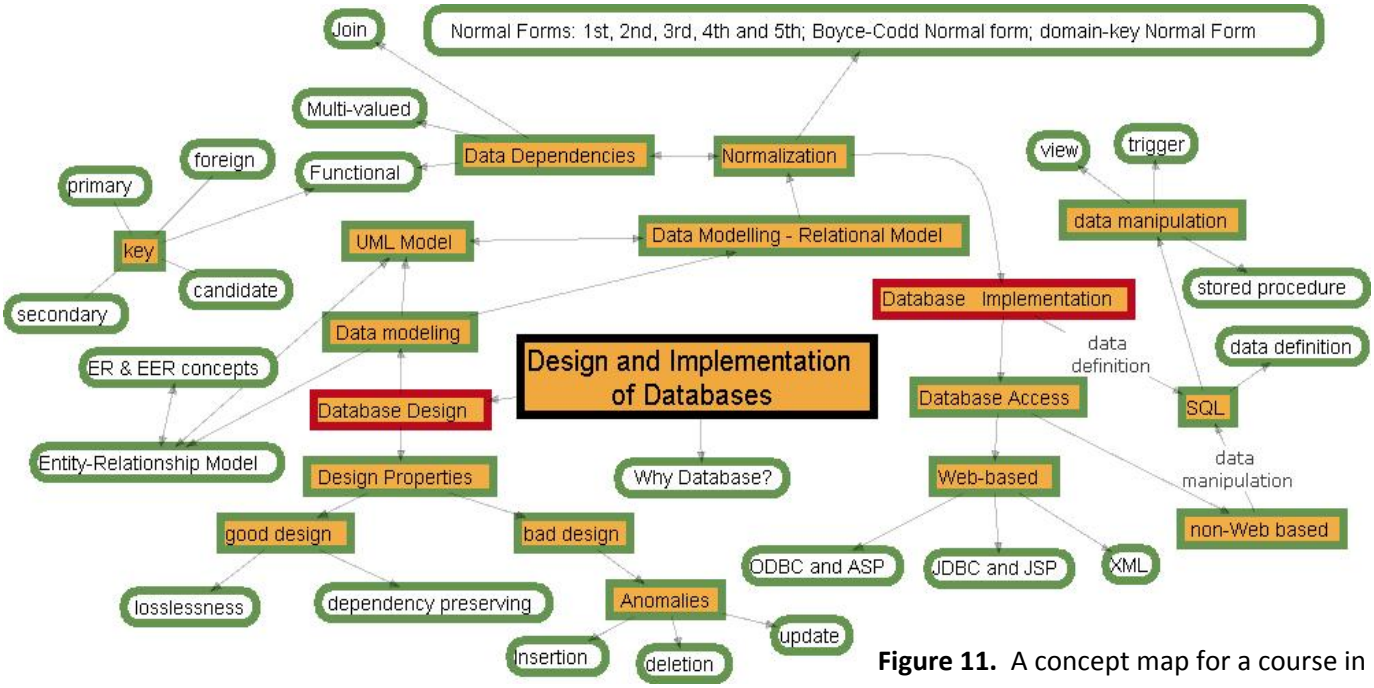

Figure 11. A concept map for a course in the Design of Databases.

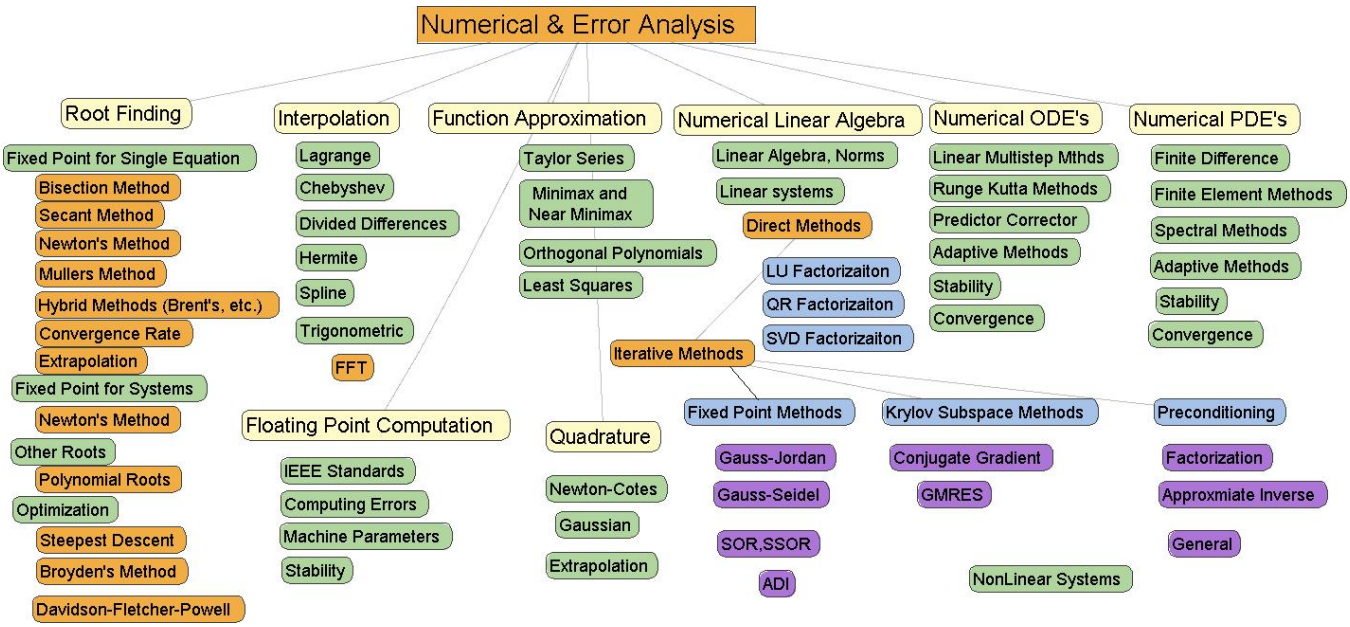

Figure 12. A concept map for a course in Numerical and Error Analysis.

Putting Pieces Together: A Sample Computational Curriculum

After two years in administrative processing, in October 2001 the Oregon State Board of Higher Education approved a Bachelor degree in Computational Physics [CPUG]. The first students entered in fall 2002, the first graduate left in June 2003, and 3-5 students typically graduate each year. Although these numbers are small, the classes are well attended by physics majors, graduate students, and engineering students. A sample of the Computational Physics curriculum is given in Table 6. It is an example of how a complete package of computation classes can be fit into a four-year curriculum that is still strong in its mother discipline.

Table 6. A sample CPUG (CP for UnderGraduates) curriculum. Computation classes in **bold**.

Year	Fall	Winter	Spring
Fresh	Differential Calculus (MTH 251, 4) Writing I, 3 Gen Chemistry (CH 221, 5) Perspective, 3	***I.* Scientific Computing 1** (Ph/MTH 265, 3) Integral Calculus (MTH 252, 4) Perspective, 3 Gen Chemistry (CH 222, 5)	**CP Seminar** (PH 405, 1) Vector Calculus I (MTH 254, 4) Gen Physics + Calc (Ph 211, 5) Gen Chemistry (CH 222, 5)
Soph	***II.* Scientific Computing 2** (PH 464, 3) Vector Calculus II (MTH 255, 4) Gen Physics + Calc (PH 212, 5) Perspective, 3	Infinite Series & Sequences (MTH 253, 4) Gen Physics + Calc (PH 213, 5) Perspective, 3 Writing II, 3	Applied Differential Eqs (MTH 256, 4) Intro Modern Phys (PH 314, 4) Writing III, 3 Fitness, 3 Linear Algebra (MTH 341, 3)
Jr	**CS Elective**, 3 Harmonic Oscillations (PH 321, 2) Static Vector Fields (PH 322, 2) Energy & Entropy (PH 323, 2) Biology, 4 Elective, 3	Perspective, 3 Waves in 1D (PH 424, 2) Quant Measurements (PH 425, 2) Central Forces (PH 426, 2) Synthesis, 3 Elective, 3	**Computer Science Elective**, 3 **CP Seminar** (PH 405, 1) Periodic Systems (PH 427, 2) Classical Mechanics (PH 435, 3) Electives, 6
Sr	**III. CP 1** (PH 465, 3) Electromagnetism (PH 431, 3) Quantum Mechanics (PH 451, 3) Math Methods (PH 461, 3) Electives, 3	***IV.* CP 2** (PH 466, 3) Physical Optics (PH 481, 4) **Computer Science Elective**, 3 Elective, 3	***V.* Adv Computational Lab** **Thesis** (PH 467, 401; 3, 3) CP Seminar (PH 405, 1) Synthesis, 3 Electives, 4 **Interactive Multimedia** (CS 395, 4)

This curriculum has been built up course-by-course since 1989 as we proposed, developed, taught, and modified new courses. The computer classes (**bold**) are seen to be distributed throughout all years of study. In total, the curriculum is a mix of existing applied math and CS classes, with the new computation classes acting as the glue that holds it together.

There is another way to answer the questions "what to teach?" and "How to teach it?" That way is to provide computation-based textbooks that help define which topics constitute proper computational education, and provide a coherent presentation of the subject. The OSU CP Education group has been trying to do that for the last 15 years. Lists of more than 50 texts and other resources (predominantly not from us!) are to be found in a recent resource letter [CP-2]. Although most of those resources and most of this paper focus on more specialized computational topics, there is still very much an open question on what and how to teach computation to beginning college science students, and who should be doing the teaching [SIGCE]. Our attempt takes the form of an *Introductory Scientific Computing* course designed to provide first and second-year students with the computational tools needed throughout their undergraduate careers, and its associated text, A *First Course in Scientific Computing* (Princeton University Press, 2005). In recognition of the widespread disagreement over which computing tools lower-division college students should learn, the paper text covers Maple and Java, while the accompanying CD contained essentially identical texts in Mathematica and Fortran90, as well as the associated notebooks, worksheets, programs, and data sets. The combination of *A First Course in Scientific Computing* and *A Survey of Computational Physics* (Princeton University Press, 2008) pave a continuous computational path throughout the undergraduate curriculum. Here are the topics covered in the first two courses:

Physics/Math/CS 265, Scientic Computing I (*A First Course*, Princeton)

OS, Basic Maple, Number Types	Logical control, plotting
Maple Functions, Number types, Symbolics	Visualization, Loops, Integration
Calculus, Equation Solving	Objects, Complex Arithmetic
Introductory Java	Web Computing: Applets
Limits, Methods (functions)	Arrays, File I/O

Physics 464/564, Intro Computational Science (*Computational Physics*, Wiley)

Unix Editing and Running*	Monte Carlo Techniques
Floating Point Errors & Uncertainties	Random Walk, Decay Simulation*
Limits: precision, under/overo ws	Interpolation, cubic spline
Matrix Computing with JAMA libe	Least-squares t, Quadrature
Differentiation, ODEs, ODE Eigenvalues	Hardware: Memory, CPU, Tuning

Online Courses and Digital Books

In addition to publishing text books, another way of encouraging the inclusion of more computation into curricula is to make at least the basic elements of computation courses available to faculty. As part of a demonstration project for establishing a national repository of computational science courses [EPIC,CSCVI], we

have produced video-based modules for our *Introductory Computational Science* course (Ph 464) [Video]. We already used them with good results in our teaching, while we are told that faculty and students at other schools are also finding them useful. In light of the previously-documented large overlap among different computational classes, the plan is to have modules cover individual topics which then can be assembled and used in a variety of classes and in a variety of schools. (A full course would require problem sets, quizzes, assessment exercises, and possibly supplementary materials in a specific discipline.)

Although we do not view the web as a good teaching medium for general education, or for students with weak self-discipline or limited motivation, it is appropriate for computational science where the best way to learn it is while sitting at a computer in a trial-and-error mode [Cornell]. Actually, the web is essentially ideal for computational science (it was invented for particle physics analysis): projects are always in a centralized place for students and faculty to observe, codes and data are there to run or modify, and interactive visualizations can be striking with 3-D, color, sound, and animation.

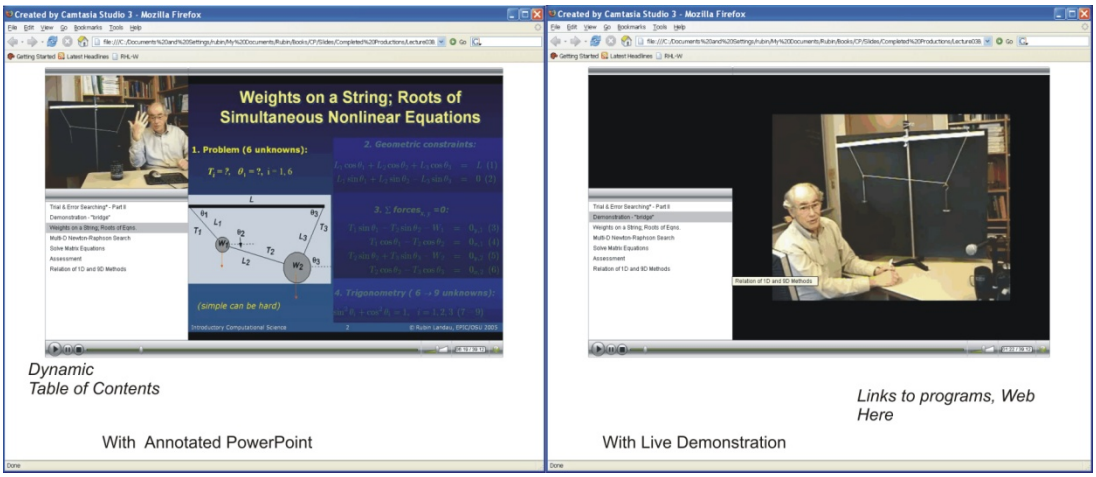

Figure 13. Two Screen dumps from a video module. The left one shows an animated slide with a speaker, the right one shows a physical demonstration.

In Figure 13 we show two screen dumps from a module on search methods for the solutions of simultaneous nonlinear equations. On the left is a discussion of the physics and algorithms needed, and on the right is an experimental demonstration of a statics problem that is simple, yet has no analytic solution. The module is viewed on a flash-enabled Web browser, and contains an "office hour" in which the professor gives an informal discussion of the material, a frame with animated PowerPoint slides, various Web links, and a dynamic table of contents.

The next step in this project is to combine our text books, web enhancements, programs, and video modules into eBooks. The MathML equations can be ported into and manipulated by a problem-solving environment such as Maple or Mathematica, or entered into search engines to find other documents with similar equations. Likewise, the figures, in *Scalar Vector Graphics* or *Vector Markup Language*, can be can be annotated to demonstrate their meaning for different types of learners, or can have their elements viewed at different levels of abstraction. Not only would this benefit disabled persons, but it would also permit any reader to use a variety of senses to understand the materials.

Summary and Conclusions

The work place has demonstrated the need to provide scientists and engineers with a better computational education. But beginnings are hard. We have seen a growing number of individuals and departments throughout the world begin to teach classes and assemble curricula in the various computational sciences. Although there is no computational science accreditation body to decide the proper content for an computational education, in practice there does appear to be a consensus regarding the basic elements of a computational science curriculum. Those elements have been tabulated and examples have been given of how they can be incorporated into courses in various disciplines.

We conclude that teaching the combination of a common computational toolset across the sciences, along with the common problem-solving mindset, is a truly liberating and attractive aspect of computational science since it permits computational scientists to understand and participate in a wider set of problems than is normal in this age of sub specialization. In addition, we conclude that an undergraduate education in Computational X based on projects and the problem-solving paradigm is more efficient and effective than a pure X education. Although students may take fewer X classes, they tend to learn the X, CS, and math better when placed in context, and thus get more out of their courses. So even if the number of X courses needs to be reduced to make room for teaching computation, this is compensated for by the increased efficiency of the pedagogy and by a more balanced view. And the motivation when students know that their education is preparing them for fruitful employment does not hurt.

Acknowledgements

We wish to thank the National Science Foundation (CCLI, EPIC and NPACI), the Oregon State University College of Science and OSU Research Office, and the Krell Institute (DOE) for support of the work presented here.

Bibliography

[AIP] *Skills Used Frequently by Physics Bachelors in Selected Employment Sectors*, American Institute of Physics Education and Employment Statistics Division, (1995); R. Ivie and K. Stowe, Physics Trends Flyer: *What's Important?*, College Park, MD: American Institute of Physics (Fall 1999).

[Cornell] P. Davis, *How Undergraduates Learn Computer Skills: Results of a Survey and Focus Group*, T.H.E Journal, **26**, 69, April, 1999.

[CP-2] R. H. Landau, *Resource Letter CP-2: Computational Physics*, Amer. Journ. Phys. **76,** 296-306 (2008).

[CPC] *Computational Physics for Upper Level Courses,* AAPT Topical Conference, Davidson College, July 2007, Amer. J. of Phys. 76, Issues 4 & 5, pp. 293-504; www.opensourcephysics.org/CPC/.

[CPUG] Computational Physics for UnderGraduates, www. physics.oregonstate.edu/CPUG/; Landau, R. H., *Computational Physics for Undergraduates, the CPUG Degree Program at Oregon State University*, Computing in Sci & Engr, **6,** March/April 2004;

[EPIC] Engaging People in Cyberinfrastructure, www.eotepic.org.

[Fink] D. L. Finkel, *Teaching with Your Mouth Shut*, Boynton/Cook, Heinemann, Portsmouth, 2000.

[Krell] The Krell Institute, http://www.krellinst.org/index.cgi.

[LEAD] *Learning through Evaluation, Adaptation and Dissemination Center*, University of Wisconsin, homepages.cae.wisc.edu/~lead/pages/projects; for reports on minorities and women, see homepages.cae.wisc.edu/~lead/pages/internal.html.

[NSB] National Science Board*, Science and Engineering Indicators*, Chapters 3-2 (1996).

[SIAM] Society for Industrial and Applied Mathematics, www.siam.org/meetings/cse07/.

[SIGCI] The ACM Special Interest Group on Computer Science Education, www.sigcse.org/conferences/.

[Survey] R. H. Landau, M. J. Paez, and C. C. Bordeianu, *A Survey of Computational Physics,* Princeton University Press, Princeton, 2008, press.princeton.edu/titles/8704.html.

[Swan] C.D. Swanson, *Computational Science Education Survey,* Krell Institute, www.krellinst.org/services/technology/CSE_survey/, (Nov. 2003).

[Terra] TerraGrid, open combinations of leadership class resources at nine partner sites, with links to other grids, http://www.teragrid.org/eot/.

[Video] *Video Lectures in Intro Computational Science,* www.physics.oregonstate.edu/~rubin/COURSES/VideoLecs/.

[VUE] *The Visual Understanding Environment project at Tufts,* UIT Academic Technology, vue.uit.tufts.edu/.

[Winch] D. Winch, *Guest Editor's Introduction: Computation in Physics Courses,* Computing in Sci. & Engr. **8**, No. 5, 22-30 (2006).

[Yas] O. Yasar, Computational Science Education: Standards, Learning Outcomes and *Assessment,* Computational Science-ICCS 2001 Int. Conf., Editors: V. N. Alexandrov, *et al.,* 1159, Springer, Berlin (2001); O. Yasar, J. Maliekal, L. J. Little, and D. Jones, *A Computational Technology Approach to Education,* Computing in Science & Engineering**, 8**, 76-81, (2006).

[Y&L] Yaser, O. and R. Landau, *Elements of Computational Science Education,* SIAM Review, **45**, 787-805, (2003); epubs.siam.org/sam-bin/dbq/article/40807.