Computing Education

Computing Research Association

Innovative Approaches to Computing Education

August 2015

The material is based upon work supported by the National Science Foundation under Grant No. (1445569). Any opinions, findings, conclusions, or recommendations expressed in this report are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Executive Summary

Traditional computer science education focuses primarily on disciplinary knowledge. The community has kept up-to-date with innovation and advancements, and has been remarkably successful in maintaining the technical pertinence of relevant curricula and has been effective in educating software professionals. With the ever-increasing integration of technology in all aspects of people's daily lives, moving forward, efforts associated with computing education must demonstrate and establish the wide-ranging relevance of our discipline and help impart the essence of computing knowledge to the general public. In the immediate term, we should focus on the diversification of our curriculum and on the needs of current and future students. It is critical to assess the needs of students in other disciplines with regard to computing so that they can be provided with the knowledge and tools they need to be successful and thrive in the workforce and contribute to society.

To this end, a computing education workshop was held in Arlington, VA, on June 12 and 13, 2014. The purpose of the workshop, which was sponsored by the National Science Foundation (NSF) and coordinated by the Computing Research Association (CRA), was to encourage participants to think about computing education 5-10 years from now. The goal of the workshop was to foster novel, transformative approaches to solve long-standing issues in computer science education.

Introduction

There is no question that the U.S. and the national economy require ever-growing expertise in the computing fields and the ability to apply computing in a host of other disciplines. The evidence is overwhelming:

- Bureau of Labor projections[1]
- Anecdotal conversations with employers of computing talent
- The near-instantaneous filling of the H-1B visa quota[2]
- Position of advertisements on various job boards (e.g., Monster, etc.)
- Establishment of subfields such as computational biology, computational chemistry, computational economics, digital humanities, and digital social science.

It is also clear that, unfortunately, the computing education community is not meeting these demands. The obvious question is, what needs to be done to change this situation? When faced with a problem, a good approach is to bring together people with a range of insights and skills and have them discuss the issues involved and to conclude with a white paper that highlights needs and opportunities. The subsequent white paper is then shared with a much broader community for their comments and suggestions. The result is that the entire community benefits and is stimulated.

The following 13 individuals participated in the two-day NSF-funded computing education workshop:

- Christine Alvarado, Harvey Mudd College
- Mark Guzdial, Georgia Tech
- Deepak Kumar, Bryn Mawr College
- Calvin Lin, University of Texas at Austin
- Darakhshan Mir, Wellesley College
- Sumita Mishra, Rochester Institute of Technology
- Kristine Nagel, Georgia Gwinnett College
- Lori Pollock, University of Delaware
- Paul Ruvolo, Olin College
- Amber Settle, DePaul University
- Kelvin Sung, University of Washington

[1] Computer and Information Research Scientists. [online]. http://www.bls.gov/ooh/computer-and-information-technology/computer-and-information-research-scientists.htm

[2] H-1B cap is reached with 'high number' of visa requests. [online]. http://www.computerworld.com/article/2907056/h-1b-cap-is-reached-with-high-number-of-visa-requests.html

- Kathleen Tamerlano, Cuyahoga Community College
- Blair Taylor, Towson University

Gary Skuse of Rochester Institute of Technology facilitated the workshop. CRA Executive Director Andy Bernat spearheaded the workshop, and NSF program officers in the Undergraduate Education Division (DUE), Valerie Barr, Jane Prey, and Paul Tymann, attended the workshop as observers. Notes were taken by both Matt Lettrich from DUE and Helen Vasaly Wright from CRA. Wright is the primary author of this report.

Pre-Workshop Questions

In order to prepare for the workshop, each participant was asked to submit a two-page "brain dump" in which they discussed their thoughts about the future of computer science education. Participants were also asked to comment on new directions and innovations in computing education. They were requested to keep the following high-level ideas and questions in mind:

1. What is the role of computer science in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for computer science and computing education?

2. What happens if we stop always thinking about and worrying about computer science recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, and generating student interest across various levels of courses?

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for computer science?

The papers were collected and their comments were analyzed (copies of the 13 papers can be found in the Appendix). A number of main ideas emerged, including:

- More computer science degree options at the undergraduate level
    - More computational-X degrees where students could train equally in computing and another discipline. Develop interdisciplinary minors that are tailored to specific computer science topics.
- Large computer science outreach study
    - Look at the various ongoing outreach efforts in bringing computation-related concepts to all levels of K-12 education.
- Make computing a general education requirement

    - Require all high schools and colleges to make computer literacy a general education requirement.
- Encourage internships and capstone projects
    - Helps undergraduate students prepare for real problems in the field.
- Computer science informs other disciplines
    - Requires baseline competency similar to math and English.
    - Encourage computer science classes to develop *understanding,* not just *use* it.

Workshop Process

Based on the brain dumps submitted by the participants, the organizers developed the following set of four questions that would be discussed at the workshop:

1. If a goal of computer science education is to prepare students to work at the intersection of computer science and other academic disciplines, ***what is the essence of computer science that should be taught to students in other disciplines?***

2. If a goal of computer science education is to prepare students to work at the intersection of computer science and other academic disciplines, ***what is the essence of computer science that should be taught to computer science majors?***

3. If a goal of computer science education is to prepare students to work at the intersection of computer science and other academic disciplines, ***what concepts and skills do computer science students need from other disciplines***?

4. Looking 5-10 years down the road, ***what are other goals for computer science education?***

The workshop opened with a plenary talk by Mehran Sahami, who had just finished serving as the chair of the ACM/IEEE-CS Joint Task Force on Computing Curricula. The workshop consisted of four sessions, one for each of the above questions. The goal was to provide an immersive environment that would engender collaborative and productive dialogues.

The remainder of this document summarizes each of the sessions. It concludes by describing a number of repeated themes that emerged from the workshop and potential next steps.

Keynote Summary

Participants were urged to think beyond the number of computer science majors and instead focus on how to make computer science accessible across all fields and at all levels. Participants were asked to think about the "Big Tent" view of computer science:

> *"As* computer science *expands to include more cross-disciplinary work and new programs of the form 'computational biology,' computational engineering,' and 'computational X' are developed, it is important to embrace an outward-looking view that sees* computer science *as a discipline actively seeking to work with and integrate into other disciplines."* [3]

Interdisciplinary computer science is becoming the norm. In emerging areas, such as data science, computer science plays a critical role. For example, genomic sequencing would be impossible without computer science. Computer science is not just a "service field" but is a vibrant industry given the significant number of computing jobs, the number of which significantly exceeds the annual number of computer science graduates. It is important to make students aware of their career opportunities and the tremendous diversity in the field.

Computational X integration is important, but tricky; it should be more than just some classes in computer science and some in X. For example, Stanford has a new computer science+X program that includes 10 choices for X and has a capstone project that shows the integration of the two fields. This program empowers students and positions computer science as both a service to other disciplines and as its own discipline. The flow of ideas has to go both ways between the fields. It should show how computer science could address problems in other fields by using real data sets and computational methods.

What is the essence of computer science that should be taught to students in other disciplines?

As we begin to think about the intersection of computer science and other academic disciplines, we need to be aware of some key elements of computer science that should be taught to students in other disciplines. This will allow those students to have a better understanding of computer science and, we hope, help them collaborate more easily. These key elements are really essential parts of the field and comprise the true essence of computer science.

[3]Joint Task Force on Computing Curricula, Association for Computing Machinery (ACM) and IEEE Computer Society, "Computer Science Curricula 2013: Curriculum Guidelines for Undergraduate Degree Programs in Computer Science," ACM, 2013.

Computing fundamentals, such as algorithmic thinking, tool use, and data storage, are essential parts of computer science that should be taught in other disciplines. These skills build off of each other; for example, algorithmic knowledge should inform tool choice. Learning about data storage and handling is important in any discipline because it allows for problem identification and simulation modeling. These are basic skills that can be applied in any discipline.

Failure is part of software system development and learning how to write code. Learning to code is not easy and students need to understand that it takes work and sometimes a lot of testing for the correct operation to occur. Often it will take several iterations of finding errors and correcting them. One bug can kill a program, but students should not be afraid to try. Failure will happen, but that does not mean students should be afraid of it. Computing educators need to do more to help students to accept that making mistakes can often times be a good occurrence.

Finally, coding skills could arguably be the most essential aspect of computer science. Code specifies the actions to be performed by a computer. It is the most fundamental part of the discipline. Learning to code should be required for all disciplines. The coding language does not matter; it is the process that is essential. The rigor of learning to code and to write working programs can be daunting to students who are new to computing. Learning to properly build something out of wood and nails can also be a daunting process. Many times those first projects have bent nails and split wood, but the result is often something of which a student is proud. More importantly, one can become motivated to learn how to hammer a nail without bending it. We need more coding classes where students can bend some nails and learn.

What is the essence of computer science that should be taught to computer science majors?

Computer science majors need to be taught everything that non-majors are taught about the field, plus they need to be able to communicate and to know how to map between computer science and other fields. Clear communication and articulation of technical details will help explain the big picture and will promote the discipline both among majors and students of other disciplines. Majors should understand the commonalities with and differences from other fields. This will allow for easier collaboration with students in other disciplines.

It is important that computer science students participate in group projects and internships that require them to work with interdisciplinary teams. So much of computer science is interdisciplinary that the earlier a computer science student starts collaborating with other fields, the better that student will be prepared for the real world. Interdisciplinary projects are beneficial for all students involved because they force them to ask questions and work together as they would in a professional work setting.

Finally, and most importantly, computer science students must be able to adapt. The very nature of this field is change. This discipline will only continue to grow and develop as more discoveries are made. The language used now will not be the same language used in 10 years. Students need to recognize that and learn how to adjust. This will require them to ask questions and keep up with the literature. Part of being a student in computer science is figuring out what is important at various levels of abstraction, learning that topic, and then moving on from it and possibly even fixing what is changeable.[4]

Concepts and skills computer science students' need from other disciplines

All students in computer science should be able to communicate their science. Oral and visual skills are important if they want to share their work with the public and promote their work publicly. A part of communication is listening. Students should be able to listen.

Students also should understand ethics and the possible ramifications of their work on people's lives. This will allow them to see the world from another's perspective. Computer science is a powerful discipline that is changing the world. Students should recognize that and know that their research could be life altering.

Long-term goals for computer science education

In the last session, the participants identified a number of long-term goals for the field that they felt that the community should strive for:

1. Enable everyone to be able to use computing as a medium for communicating and exploring ideas.
2. Reach a place in general education where computer science is seen as a fundamental subject in K-12 and the undergraduate level.
3. Educate students so they can become developers who can build robust, scalable, and secure systems.
4. Popularize computer science, through television shows or books, so that the broad population knows what it is. Currently, it has no mass media presence and is not well identified by the public.
5. Learn what not to do from other disciplines. Do not repeat their mistakes.

[4] Note there was no future curriculum discussion about what computer science majors have to have. This is perhaps because the key questions, with their strong focus on interdisciplinary work, biased the nature of the discussion.

6. Produce qualified K-12 computer science educators.
7. Make computer science a required course for all college majors.
8. Make computer science a prerequisite for college admission.
9. Finally, increase the field's diversity.

Observations

A reoccurring theme among workshop participants can be summarized by the phrase "get over yourself." Computer science is intellectually challenging and practitioners tend to be intelligent and driven. Unfortunately, these characteristics lead others outside of the field to view computer scientists as difficult to approach. We collectively need to overcome this perception so that productive collaborations can be established for the direct benefit of our students, both within computer science and in other disciplines that would benefit from components of computer science in their curricula.

- What is computer science?

  The answer is not as simple as it seems. The computing landscape is constantly changing and evolving. Does computer science have a number of subfields, or is computer science a subfield? When computing degrees first appeared, they were called computer science. After the field matured a bit, it became clear that a new field was emerging, namely information technology. What new fields have appeared since then? Is a field, such as bioinformatics, part of computing?

- Failure is an inescapable essence of computer science.

  Failure is part of software system development and learning how to write code. Computer science students, and even practicing computer scientists, should welcome the possibility of failure and not be afraid to test their systems. Learning how to write code in different programming languages is challenging, and failure at the beginning is not only inevitable, it is an important part of the learning process. Despite these experiences with failure, it is vital to learn how to code because coding is the key for data representation. Coding is not only a practice, but it teaches logical thinking and serves as a building block for advanced study in the field.

- Algorithmic thinking should inform the choice of programming tools.

  Good, robust design principles can lead to sound software engineering practices and the development of innovative approaches to solving problems. Students need to know how to ask questions in order to understand the problems that they are trying to solve. They should have the ability to work at different levels of abstraction and be able to debug their code at these different levels. Specific projects that target Human-Computer Interaction (HCI) and domain specific languages force students to ask questions through the nature of assignments. In addition, those projects can teach them about using existing tools rather than always starting from scratch.

- A culture change in computer science is necessary.

  A significant challenge that needs to be overcome in computer science is its culture. Computer science does a poor job of attracting students and, therefore, slow progress has occurred in computing education. A shift in the culture of the discipline needs to happen. Instructors who teach the discipline must care more about the education of our students. They need to adopt teaching practices that have been developed in other sciences. The emphasis should be placed on education rather than on the science itself. In support of this contention it is important to note that there are no computer science education departments in this country. Very few professors are known as both computer scientists and computer science education researchers. Changes in other disciplines, notably biology and physics, which drove the shift to more research-and evidence-based student engagement practices, are not present in computer science. As a group, computer science educators must broaden their teaching approach to include best practices developed in other disciplines.

  It was noted at the workshop that a culture change was particularly necessary at the large universities with the potential to reach hundreds or thousands of students. These universities produce top-class research projects and perform cutting-edge computer science, yet their curriculum and pedagogical approaches often lag years behind their small-college counterparts because of the class sizes at these large institutions and the lack of emphasis on teaching.

- Given the skills to enhance education, a culture change can occur.

  New pedagogy courses could be developed for faculty members that focus on change and on a community of practice. Faculty development workshops could be established for both computer science faculty and domain-specific faculty that will encourage collaboration and the exchange of effective teaching methods. These workshops could serve as prototypes of interdisciplinary collaboration for actual courses. Quite a bit of information could be learned from the initial cohort of teachers who participated in the first round of classes and workshops that could be share with later cohorts. This would make pedagogy courses stronger and the computing education community larger, more collaborative, and potentially very effective.

In addition, large universities could adopt some of the practices of smaller colleges to better engage their students and improve their pedagogy. Although classes at large universities will never be small, TAs and undergraduate tutors can provide a more personalized experience for the students, making them feel more like part of a community of learners, and provide them with the necessary support during their (expected) struggle to learn computer science.

- Students need to change, too.

  Computer science majors should be encouraged to engage in issues with social impact and with projects that include and consider ethical issues. As computer science projects become more interdisciplinary, students from other fields will be introduced to a wide array of computer science-related topics. This will open up departments for collaborations or lead to the development of dual degree programs. A more welcoming environment in computer science departments could also lead to increased female and minority student enrollments. With these changes, there will be an overall adjustment in the culture of computer science.

- Computer science needs to become better understood.

  The public does not yet understand the discipline, and we do not do a good job of educating them. Computer science does not have the appeal of natural sciences such as biology or physics because people do not experience it, or realize they experience it in their everyday lives. The field needs to be more accessible so the public can be educated about the work that is performed in such an important specialty. This change may need to start with an educated electorate. By educating elected officials we may see a shift in the opinions and attitudes among the public concerning computer science as a field. An increased level of interest in the discipline may even arise from such a promotion of the field.

- More breadth in computing education.

  The computer science degree needs to be diversified. Students should be required to take classes in other disciplines to expand their general knowledge and to open up opportunities that can be related to both their degree and other areas of research beyond the bounds of traditional computer science. Every student should be required to take a computational-X course or be encouraged to minor in another discipline.

The consequent integration of two fields will empower students both during the pursuit of their undergraduate education and thereafter as well. They will be better graduate students, better members of the workforce, and more informed citizens. The additional educational requirements of the discipline will make them well-rounded individuals. It will provide a context for computing. Programming will become a means to achieve an end, not the end itself. Students will still need rigorous software engineering skills, which they will learn in their computer science classes. However, as more degrees require some small aspect of computation, they will foster the introduction of disciplinary breadth into the field and make the discipline more accessible to outsiders.

Computing should be required.

All students, regardless of their major, should be required to take computer science classes. One or two courses in computer science should be as commonplace as requirements for English, math, or history. This would expose other students to the discipline and make the subject more accessible. Another way to make computing more attainable to the public is to popularize it. YouTube videos about core topics that are engaging, memorable, and accessible to those outside of computer science can bring difficult-to-understand concepts to the attention of people outside of the field and minimize the perception that computer science, and computer scientists, is unapproachable. Another possibility for popularizing computer science could be to find a renowned science writer to create a piece of literature on a computing topic that would be attractive to the public. However, this last approach is likely to be fraught with uncertainties that negatively affect the probability of success.

Potential Next Steps

The conversations that were initiated at the workshop need to continue. Colleagues in other disciplines, particularly non-STEM ones, need to be included so that together we can identify aspects of computer science that are needed in their curricula and identify interdisciplinary opportunities for collaborative projects for our students. This is only the beginning of the discussion.

Appendix

Responses to Questions: Christine Alvarado (Harvey Mudd College)

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?

   I must admit that when I first read this question I had to look up the difference between multidisciplinary and interdisciplinary. So I am first going to start with a definitional statement, just to make sure I am understanding things correctly.

   I view what this question is asking as about the shift toward creating new computational science and computational arts/humanities fields, which are somewhat separate from computer science. While in the past one was a biologist, who might also know a little computer science, now the idea of being a computational biologist is becoming commonplace. However, I don't think that these shifts necessarily change the core of the discipline of computer science. I will talk about this more below in responding to question 3.

   In terms of how these shifts impact education, I think there is an opportunity to shape how CS is taught and viewed in all of these interdisciplinary programs, but also (and perhaps more interesting), an opportunity to use these interdisciplinary programs to try to tease out what the real essence of computer science is by looking for the intersection of how CS informs and is used in all of these different interdisciplinary fields. What exactly are the skills and knowledge needed from CS in these other disciplines? How can we teach these skills effectively in the absence of all the other skills and knowledge that come with a CS program? If we can do so, might we also make our early CS courses more targeted and effective?

   There are also many challenges, of course. The two most pressing I see is that (1) if we are not strongly involved with education in these emerging fields until they are well-established, we risk that the CS portion of the education in these fields will be shallow at best (or misguided at worst) and (2) conversely, if we try to heavily involve ourselves in all of the emerging interdisciplinary fields, we run a serious risk of spreading ourselves way too thin.

2. What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number

of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?

> Well, I am not sure we should ever stop worrying about these issues, and I am not sure I understand the difference between a focus on recruitment and retention vs. a focus on exposure, broad outreach, and butts in chairs. The best I understand this question is to say that what if we stop focusing our efforts on getting and keeping individual students, but instead just try to make the field more exciting and let more people know about it (and then assume this will result in getting and keeping the students).
>
> Assuming that my understanding is correct (and again, I might be way off here), I think this shift to some degree might be good. While I think we should be sensitive to students' needs and issues, I worry that at the moment we're catering a bit too much to student interests. In particular, the emphasis on games at the early level is a bit overwhelming, and I'd like to see a little more diversity.
>
> I like the idea of trying to focus on what makes the discipline truly exciting independent of what the current fads are. I think the discipline and its applications have matured enough that students are starting to understand that CS is useful for many things they care about, but not how CS is useful. I think we can start to think about how the big ideas in CS (e.g., what is data? How can you work with it? Why does efficiency matter? What logic is needed to write programs? etc.) can be showcased and tied in not specifically to the latest fad that students are thinking about (or, rather, games), but for larger challenges and ideas that they are now familiar with (e.g., searching for data, visualization, communication). Of course, figuring out how to tie the ideas to their general utility is something I don't really know how to do, but I think it's a compelling lens with which to think about education.
>
> However, in all this, I think we generally need to be careful not to ignore the issue of the culture of the field of CS. In fact, even if we stop directly targeting students and think more about ideas, I think we should probably think more about the culture of our community and inclusion in our community and make sure that it's as inclusive and broad as possible.

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

> I think that CS should play a huge service role, just as math does. However, I am wary of embracing the "majors" vs. "non-majors" classes model until we

grow a bit as a field and can better understand the core ideas that will be of service to other majors/programs. In math, the difference between majors classes and non-majors classes tend to be the depth of study and sometimes the specific organization of material (e.g., math for engineers). On the other hand, non-majors courses in CS seem to be fairly fundamentally different from non-majors courses. The latter tend to focus on applications (e.g., Excel, HTML). I think once we've better identified the core ideas from CS that serve students broadly in other disciplines, the CS non-majors courses will actually look more like majors courses, except perhaps in their application area and some depth of study, sort of the way it is in math.

Of course, we also need to be clear about delineating what the core CS topics are for the major and not lose those tracks in our shift to becoming a broader, interdisciplinary, and service-focused field.

Responses to Questions: Mark Guzdial (Georgia Tech)

Paul Rosenbloom argues that computer science is not just a science like chemistry or physics, but a whole branch of sciences: "The Computing Sciences" (http://ubiquity.acm.org/article.cfm?id=2590530). In his book On Computing: The Fourth Great Scientific Domain, he argues that the interaction between computing and other domains creates a whole set of sciences, on par with the physical, life, and social sciences.

Rosenbloom is talking about the study of computing and how it interacts with other studies. That scholarship importance for advancing all of the computing sciences, in an interdisciplinary manner. We might broaden our scope and consider the ways that computing influences how everyone works and communicates. I do believe that eventually we will be teaching everyone computer science.

I was inspired to work in the field of computing education when I first read Alan Kay and Adele Goldberg's 1977 paper, "Personal Dynamic Media." Kay and Goldberg paint a compelling vision of computing as a medium, as a way of expressing and communicating thought. They explain that computing goes beyond other media—it's a meta-medium, in that a computer can replicate and combine all other media, plus it's active and can respond to the reader.

In Kay and Goldberg's view, we shouldn't be teaching computing to everyone in order to give him or her vocational skills. We are teaching them computing to give them a voice, to let them say things and communicate ideas and in so doing, think and problem-solve in ways that they would not otherwise be able to. Andrea diSessa called this use of the computer as a communication/thinking tool "computational literacy" many years ago, decades before we started talking about computational thinking.

Eventually, we can expect computational literacy to appear in schools like textual literacy (reading and writing text) and numeracy (mathematical literacy). We use these existing media literacies across the curriculum. We read and write language, and use mathematics in every science, engineering, and business course—and in almost every other discipline. We also take courses explicitly in the medium (e.g., in language, composition, and mathematics), because we want to develop our understanding and facility with the medium, not just use the medium. We will also need computer scientists and software developers, people who specialize on the medium and the study of the medium. But like in English and mathematics, these will likely become a minority.

We are passing the point where only the computing specialist reads and writes the computing medium. Some estimates suggest that for every professional software developer, there may be as many as nine other end-user programmers who use computing as a medium for problem solving and expression. But the computing medium is still written by a minority, by those privileged to have access.

Universal literacy is always hard to achieve. Illiteracy rates are still high in many parts of the world. Many wish that mathematical literacy (numeracy) was higher in the U.S., because better understanding of mathematics might avoid some of our economic crises.

We’re still at the early stages of making computing available to everyone. To achieve universal computational literacy, we need teachers, curriculum, research into effective ways of teaching, research into how students learn computing, and more experience with providing computing education to a broad range of students. As we move toward universal computational literacy, we will undoubtedly further all of Rosenbloom’s Computing Sciences. More broadly, we will give everyone a new tool to think with.

(Portions of this essay are drawn from my June 2014 Blog@CACM post.)

Responses to Questions: Deepak Kumar (Bryn Mawr College)

Current Situation: Some Observations

Over the last five years, most computer science departments have seen an increase in enrollments across the board in all courses. The last two years have seen unprecedented growth in enrollments, which, by some accounts, has surpassed any prior enrollment growth periods in the history of the discipline. Here are some specific observations:

1. Students are flocking to introductory courses in unprecedented numbers

   On most campuses across the country, 10% (or higher) of the entire student body is enrolling in introductory courses. At select campuses (like Stanford, Harvard, MIT, etc.) this number is as high as 90%. Yet, most universities (with the exclusion of those mentioned above) will award only 5% of its degrees in computer science, at best.

2. Second courses

   There has also been tremendous growth in the enrollments of second courses. While enrollments in these courses may have doubled (or more), the growth is not quite as dramatic as in the first courses. Still, not all the students who take the second courses will end up being computer science majors.

3. Increased (non-majors) demand for upper level courses

   Consequent to (2) above, several upper-level courses are also experiencing robust growth. The growth, when observed closely, is most dramatic in courses directly related to the areas of job growth in the tech sector (see below). That is, no so impressive in compiler courses, but truly dramatic in machine learning courses.

Why? Some "explanations."

In my mind there are two primary forces acting here.

1. The economy

   Since the crash of 2008, there has been a drastic decline in the demand for students pursuing majors in economics and business. Many of these students are now seeking degrees in computer science. Moreover, computing jobs account for nearly ¾ of all the jobs in STEM fields in projected jobs growth estimates in the next five years. The ubiquity of apps, their conception, and development requires skills that can only be acquired through a deeper study of computing and yet, it represents a huge bulk of the growth in the tech industry in the coming years. Many students are flocking to computer science for these reasons.

2. The role of data & information

   Many students in other disciplines are recognizing the role of data and information (and hence the modeling, analysis, and its understanding) in their areas of study. Beyond the STEM disciplines, even the humanities and the social sciences are experiencing the importance of the emerging role of data. While the term "Big Data" may be an overused industry term, there is no doubt that the emerging areas of "data science" (aggregation, modeling, mining, visualization, etc.) and machine learning have already shown tremendous potential, promise, and applications. Universities with course offerings in these areas are showing the most demand from non-majors in upper-level computer science courses.

Questions Posed

1. Role of CS in truly interdisciplinary efforts versus multidisciplinary efforts. What are the implications, challenges, and opportunities for CS and computing education? Butts in chairs, etc.

   Interdisciplinary efforts, more than multidisciplinary, require computer scientists to be experts in domains beyond computer science, and also require those in other disciplines to be more proficient in computing. The current structure of offerings by most computer science departments does not meet either of these challenges.

   It is also not clear that there needs to be additional formal structures created to address these needs (like creating new departments: Digital Humanities, anyone?). However, with a creative rethinking of the curricula and, more importantly, the structures for students to obtain interdisciplinary credentials, I feel that there are ways to address the current situation for the near future. Here are some possible courses of action:

   Curriculum level:

   Design pathways through the computer science department's offerings for students to specialize in their domains/careers of choice: software engineers, systems, networking, business applications, bioinformatics, digital humanities, data mining, etc. That is, a department should (1) allow students

to design or choose pathways toward some specialization in computing degrees (beyond software engineering), and (2) create new and innovative course offerings in the emerging areas. The latter could be done in collaboration with faculty from other departments.
Some (few, not many) universities have experimented in the recent years with the creation of formal "Streams" or "Threads" in their programs. Perhaps we should evaluate their success and provide means for adoption of best practices.

Beyond the threads/streams that lead to a degree in computer science/computing, there is still the need to provide credentials to non-majors who are taking computer science courses. The typical approach to this is the minor/concentration in computer science. However, requirements for these still focus primarily on computer science foundations, rather than addressing the interdisciplinary aspects.

Create new minors in computing for non-majors that can be tailored to the motivations coming from outside computer science. For example, at Bryn Mawr, we have created a minor in computational methods that, besides an introductory set of courses in computer science, requires students to take a couple of computational courses within their home departments. Such a structure provides clear pathways for students to develop foundational interdisciplinary skills that will be relevant to their primary fields of study. This approach also promotes better integration with faculty in other departments and computer science (that computational physics course, offered in the physics department, requires background in computing that comes from the introductory computer science courses).

Course level:

Change the content of lower-level computer science courses to reflect and cater to the new interdisciplinary perspectives. That is, focus more on data and information, its storage, acquisition, modeling, analysis, visualization, etc. Needless to say, examples should come from different domains. More specifically, focus in these courses should be on various ways of handling data and NOT learning to design the data structures, most of which are provided in libraries in most modern programming languages. Moreover, the use of specialized APIs, their design, and perhaps even the design and use of domain specific languages is going to play a bigger role.

This is different, yet builds upon, the so-called "contextual" approaches to teaching introductory courses. For example, the use of robots, media-computation, scientific computing, graphics, arts, etc. While each of these approaches are attractive and have shown tremendous promise in attracting students, they also suffer from a single-minded commitment to one context! No matter how engaging or interesting the context, there are always a sizable percentage of students who are averse to engagement ("Robots are not for me," "I am not an artist," etc.).

2. What happens as we mature as a field? Service roles? What road do we envision for CS?

   This is a big one, but I think I have laid out a near-term sketch of what could be considered to possibly address some of these concerns. I think the service role (beyond the students who take introductory courses to fulfill the math requirement) is outdated. Every role (thread/stream) is important and hence it must be identified and properly integrated. This is what truly interdisciplinary means, right?

Responses to Questions: Sumita Mishra (RIT)

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?

My related experience:
Currently, I am working on developing curriculum in critical infrastructure protection (CIP) across different education levels (high school, 2- and 4-year colleges), and across multiple disciplines (computing and non-computing). In order to bring in the cyber, physical, and human aspects involved in protecting our nation's critical infrastructure (ranging from communication to transportation to healthcare), the CIP concepts are developed and embedded in curriculum across different institutions and different disciplines. With CIP being the common theme across these courses that are spread across disciplines, it makes this effort truly interdisciplinary.

Implications and Opportunities:

- Embedding computing concepts in courses across several disciplines and different education levels broadens the knowledge base and creates a more "cyber-aware" workforce.
- Interdisciplinary effort involves interaction between computing and non-computing faculty and generates opportunities for collaborative research. Collaboration among faculty in 2- and 4-year institutions provides opportunities for broadening faculty expertise and knowledge.
- These efforts can help in teaching students "how" to think rather than just educating them about "what" to think. There is a demand for a trained workforce equipped with critical thinking and problem-solving abilities, and the ability to adapt to changing times. This cannot be achieved by just teaching the canned computing techniques, without bringing in concepts from other disciplines.
- Interdisciplinary efforts help in extending the acquired knowledge to practical and innovative solutions. It would also help in imparting the material more effectively to CS students (we are all comfortable delivering the technical material and the concepts from other disciplines can take the back-burner, which can be sometimes detrimental).
-

Challenges:

- Most universities have very rigid departmental structures that can sometimes be very difficult to break.
- It is not always easy to accept the implications of other disciplines on computing education. For example, a major component of critical infrastructure protection is the human aspect. Hence to bring in this aspect in

CIP education, a collaborative effort between psychologists and computer scientists would be ideal. However, a major challenge is the willingness of faculty to pursue such collaborations. Acceptance of discipline-specific language from other disciplines can be difficult.

- Sometimes dissemination of the results of this effort can be difficult. For example, if the cyber aspects of CIP are embedded in a business course, what would be the right venue to disseminate the findings of this effort? Will it be a good fit for a CS conference?

2.What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?

- Maybe we can stop worrying about CS recruitment and retention in general, but it is very difficult to not worry about gender-specific and underrepresented groups-specific recruitment and retention issues. Based on the number of female students and students from underrepresented groups in our computing classes, this is a problem that is a growing concern and cannot be ignored. Broadening participation in computing is essential in my opinion and targeted efforts are required in this direction.
- The efforts in broadening participation and outreach need to begin much before the college years. The pipeline in computing education needs to be created, starting at the elementary and middle school levels.
- Students can relate to certain computing topics (e.g., social networks, cybersecurity) at a very early age. Hence, if the high school programs are geared toward these topics, maybe it will help in attracting a more diverse pool of students toward these topics in particular and the computing discipline in general.

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics, we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

- We should be aware of the market as it evolves. One thing that is constant in the CS field is change. CS education has expanded considerably over the last several years to include several evolving concepts. For example, cybersecurity has traditionally been a sub-area in most CS or Engineering curriculum. However, RIT launched the department of computing security in 2012, with both BS and MS degrees offered in this sub-discipline of computing. Similarly, a few years ago, undergraduate and graduate degree programs in game design were launched to address the needs of this sector within computing. Many of the core CS concepts are still integral parts of these degree programs. However, the advanced concepts are developed based on the growth in the range of applications. Hence, we have to constantly evolve, while maintaining the core structure of our programs.

- As we move toward more interdisciplinary efforts, computing will be more and more embedded across multiple disciplines, rather than being a standalone subject matter. There are certain core computing concepts that should be a part of the curriculum across all disciplines. This could be done by offering CS courses as part of the general education curriculum or by embedding certain key concepts in existing courses across disciplines.

Responses to Questions: Kristine Nagel (Georgia Gwinnett College)

Computing baseline and gen ed requirement for all: Computing education will be included in the general education requirements for all colleges, for all majors or degrees. This is not just an introductory course, but requires a baseline competency similar to math and English, so more depth in computing will be possible for all. Assume college-ready student has a level of computing literacy, but will add skills and tools to support more complex problem solving. The student will expect their investment in learning to provide additional resume items and opportunities. If the requirement is a two-course sequence minimum, envision the students seeing the variety of computing disciplines, just as social sciences encompass a wide range of well-known disciplines such as anthropology, sociology, and ethnography. Undergraduate students will view computing similarly, as a diverse set of studies, that taken together account for the breadth of computing contributions. College students will be able to learn about artificial intelligence, human-computer interaction, algorithms, operating systems, simulation, networks, security, and software engineering, not just about using personal productivity tools or how current enterprise systems operate. With a full year of computing studies in one such area, a student would be able to learn the basics and how to apply them in problem solving, design choices, and extending beyond current solutions. This could be part of the preparation for internships and/or service learning early in the degree program.

Computing learned within real disciplinary problems: A gen ed computing sequence emphasizing depth of knowledge in a specialized area and problem solving is a natural fit to leverage data and problems of another discipline, one that interests the student enough to engage them in an ongoing effort to develop a solution, requiring computing knowledge. Computing experts will need to work with the discipline experts to apply tools and processes, not just to share these techniques, but to do the collaborative work of applying technology to develop real-problems in the field, whether from actual "cases" or "in-the field." A faculty member will actually live the mantra of lifelong learner–in another discipline. The apprenticeship or contextual learning model has already been applied in software engineering, as well as introductory programming. Could contextually relevant studies increase students' interest and effort to apply simulations to determine best "what if" cases and to analyze data to answer real questions? For instance, within your area of interest, what are the processes used and where is there opportunity to improve workflow? Relieve humans of repetitive activity? What are the sticking points? How can technology be applied to develop potential improvements, or what would need to be changed with existing computing to make a difference? Developing and implementing solutions to real-world situations is engaging for students, but an ongoing challenge for teachers.

Faculty excellence: In addition to the close collaboration to learn another discipline, there is also knowledge about the existing learning tools and when they are effective or not. The faculty is not only the creator of curriculum, but also curates the increasingly diverse and vast collection of learning materials, assessment instruments, and self-evaluation tools. Will U.S. colleges place the most experienced and best faculty in these key undergraduate instructional sequences to provide the best experience, to those who most need to learn? Or will computing faculty advocate for the best to be teaching the entry-level courses?

Motivation vs. pedagogy: How will we use analytics to assess success and barriers? If ready for studies, then how do we provide credit for prior experiences, whether in classroom or not? What does the "credit hour" actually represent that correlates to value either in the workforce or in preparing for the future?

Student experience vs. content delivery: Delivery ranges from face-to-face, hybrid, online, MOOC. But student experience is the other value in the equation—immersive/residential, commuter, virtual attendance, asynchronous study. The concept of credit hour as seat time lacks a real-value to either the student for learning or the employer for skills gained. The synchronous and assembly-line set of semester-long courses adding up to a college degree no longer fits, as more adult learners seek to add computing skills to further their careers, but must fit the learning between work and life responsibilities. New choices for skills and professional development vs. learning the foundations for the future

Responses to Questions: Lori Pollock (University of Delaware)

Please send a 2-page brain dump in which you discuss your thoughts about the future of CS education, keeping in mind these high-level ideas, as well as other high-level concerns that come to mind. What sort of computing education activities, what new directions, what innovations come to mind?

As we begin to think about computing education 5-10 years from now, there are a number of high level ideas/questions that arise:

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts. As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?

CS could provide the coordination and coherency in interdisciplinary efforts through conversations and challenges around data and cyber-physical systems. Many disciplines can work more closely by bringing together their data to integrate knowledge more closely than working separately with only certain parts of the data, or using others' data only within their own perspective. CS has the opportunity to help create and improve interdisciplinary efforts through data and methodologies to extract knowledge from that data. This is different from the stereotypical role of providing software for other disciplines. There are more and more opportunities for artificial intelligence, data mining, information retrieval, and robotics, cyber-physical systems with software and sensors, etc., to take precedent in solving interdisciplinary problems, whereas our traditional curriculum does not stress these CS areas.

Some challenges for CS education is moving toward creating more opportunities for interdisciplinary projects where CS students work alongside students from other disciplines, learn the terminologies of each others' disciplines, appreciate their problems, and work toward solving problems together. Similarly, CS students need to gain a good grasp of the different kinds of computing-based problems in other disciplines, and more of these methodologies that can drive these efforts.

Similarly, non-CS students need to have more appreciation for the potential of data and computing and what can be possible, so they can think outside the box with that general background of CS principles.

2. What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?

I believe exposure and broad outreach means providing more access to the CS principles and how CS impacts individuals, organizations, and the changing world. This is less about creating more computer scientists, and more about getting more of the general population exposed to CS concepts and thinking computationally in their own work. To me, this means not focusing so much on programming in the first few courses, but broadening that course sequence to other aspects of CS, with programming scattered throughout the major curriculum as needed. Many students see themselves doing some form of computing, but not necessarily programming and becoming super proficient programmers, which seems to be what most of our first two-year students are focused on before they see all the various areas of CS that lead to more interesting concepts and implications and interdisciplinary work.

To broaden the participation in computing by students in other disciplines, more CS courses could be created or modified to not rely on heavy programming to understand and use the concepts. For example, a data mining course that uses Weka instead of programming: Does it require two years of programming courses before taking it, which would bar other majors from getting exposure to those concepts and tools? Similarly, a software testing course is useful to many disciplines where computing is involved, but maybe does not require two years of programming courses before taking it?

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

I see different paths that should be provided through a CS department. Some paths are for those who are very interested in systems level, architecture level CS, creating the systems we need for our machines to support the wide variety of computational activities. There could also be paths that are more data-oriented CS with the focus more on getting knowledge from large data of various forms and from different disciplines. There could be paths that are more directed to creating and supporting cyber-physical systems where robotics, sensors, reactive or adaptive systems, artificial intelligence kinds of courses are involved. I believe the paths need to start wide at the start so students can figure out which path they like most without losing ground in their studies, but also provide exposure to the wide variety up-front so they can choose early. Computational science paths.

We do not want to become the software producer for other disciplines where the other disciplines are just clients. We don't want to just produce programmers or teach programming to the masses in a service role. We could use interdisciplinary themes and exposure as drivers for new paths and starting points to courses at different levels.

In summary, my experience has shown that getting CS students in a room working with non-CS students provides challenging, fruitful, and enlightening team project work. These kinds of interdisciplinary project courses could be created on different paths, at different levels, all the way down to the introductory level using problem-based learning, team-based learning, and classrooms with students from different disciplines.

Responses to Questions: Paul Ruvolo (Olin College)

Interdisciplinary → Multidisciplinary

Challenges:

- **Iteration + collective faculty ownership:** The courses at Olin that are both truly multidisciplinary and successful are all courses that have been taught many times by many different faculty. Taking advantage of having not just one person from each relevant discipline, but multiple people, has been paramount to these courses succeeding.
- **High-level messaging and conceptual framework for course content:** Students have been conditioned throughout their K-12 education to think of things in disciplinary boxes. Without doing some explicit work to replace their conceptual framework for content, it is easy for a multidisciplinary class to be confusing, disconnected, or inaccessible to students.
- **Downstream connections in the curriculum:** Similarly to the point on conceptual frameworks, to maximize the impact of a transdisciplinary course there should be reinforcement of the material and content presented later in the curriculum. Traditional disciplinary courses (and even interdisciplinary ones) tend to have many opportunities for their content to be reinforced throughout the curriculum. We must replicate this if the transdisciplinary approach is to prove successful.
- **Providing appropriate coaching to students:** Given the newness of transdisciplinary approaches, it may be challenging for faculty to provide high-quality advice and direction as students tackle open-ended projects for these courses. As a field, we will want to develop best practices and guidelines to address this issue.

Opportunities:

- **Radically rethinking college-level math:** Modern computers and their rich software suites have become the perfect platforms for learning college-level math in an engaging, project-oriented fashion. Further, the flexibility of teaching math using a computational approach provides the instructor with many choices as to what level of detail to teach the material, and then making things more or less concrete by either building on or peeling back the computational layers of abstraction. We must not miss this opportunity to have an impact on the pedagogy of mathematics.

- **Computing as a toolkit for everyone:** The pool of people that should have some knowledge of computation and how to harness modern computers to solve interesting problems is ever increasing. As computer science educators, we have the opportunity to make this case in a persuasive way to a broader range of students through developing new transdisciplinary courses.
- **Data science as an early integrative experience for students:** Data science is the hot buzzword of the day and, so far, most of the innovation has been happening in the space of elective courses that more senior students take. I would like to see us, as a field, explore the potential for data science to be a cross-cutting content area that can be used to develop a transdisciplinary course that a wide swath of students takes early in their undergraduate careers.

Shifting focus away from recruitment and retention

I think that there is an opportunity to make a conceptual shift from evangelizing computer science as a discipline and instead focusing on evangelizing computational approaches. The goal should not be to make everyone a computer science major, but to convince as many students as possible that adding computational methods to their arsenal of problem-solving techniques will make them a better student in their major, whatever that may be. However, I do think that this shift has to be met with a similar open-mindedness to computational approaches from the faculty in other disciplines.

If such a shift in focus were made, we would also have to think crucially about how issues of increasing participation for underrepresented groups interacts with this shift. It seems likely that the shift could positively impact this aspect of computing; however, we must engage with it in an intentional manner.

What will happen as we mature as a field?

I firmly believe that computing and computational approaches will become something that more and more students learn about outside of the context of being computer science majors. I don't think that this should relegate us to a "service" role. If we think about what we bring to the table for students in other majors, it is not just how to program, but how to think about hard problems from a computational point-of-view. We are uniquely qualified to instruct students in this type of thinking.

Miscellaneous brain dump: Some of these may provide good fodder for discussion.

- How can our courses both inform and be improved by extracurricular computing activities?
- What are the areas of opportunity over the next 5-10 years when it comes to students learning from and teaching each other? How can we harness their excitement?

- How do physical work spaces interact with the computing curriculum of the future? If we assume that transdisciplinary is the way to go, how can our computer labs (if we even have such a thing) change to better support this?
- How do we redesign the way we teach data structures to deal with the living in the age of big data? How do we move from storing data to answering questions with data as the fundamental focus?

Responses to Questions: Amber Settle (DePaul University)

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?
2. What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?
3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

Some interesting ideas about the future of computer science education come out of examining the role mathematics currently plays with respect to other disciplines. There are few disciplines where math plays no role whatsoever, and in some disciplines, such as economics, the social sciences, and medicine, the field would simply not exist in its current form without mathematics. At the same time, the broad applicability of mathematics in other disciplines has not restricted the innovation that mathematicians have achieved in their own field. Mathematicians have instead found new inspiration in problems taken from other disciplines, and the field, as a whole, is flourishing.

I argue that the potential of computer science will only be fully realized if the field is able to approach the same level of involvement in other disciplines as mathematics does. This is clearly more difficult for computer science since it is a significantly younger discipline that is not firmly entrenched in education from the earliest years the way that mathematics is. The outreach of computer science into other disciplines has also been hampered by the perception that overtures by computer scientists are self-serving. Computer scientists do not make friends when they approach other disciplines as outsiders and try to show practitioners how they can improve what they do by using computer science. This stance is justifiably seen as egotistical and arrogant.

One way for computer scientists to begin to achieve significant integration into other disciplines would be to produce students who understand multiple disciplines in deep ways. This is not a new idea, since areas of inquiry classified as computational-X for some value of X have been of increasing interest in the past decade. However, researchers in computational-X are typically either people holding academic positions or graduate students studying with those academics. While these researchers have certainly produced innovations of note--for example, in computational biology, computational finance, and computational linguistics--

they remain a relatively small population. The potential for innovation in any computational-X field would be increased if larger numbers of people trained equally in computing and another discipline existed, and if such people worked in areas outside of academia. This could be achieved if computational-X programs were more common at the undergraduate level, where larger numbers of students are to be found and where the people educated by such programs are more likely to seek jobs outside of academia. While it can be argued that undergraduates can achieve deep knowledge in multiple disciplines by double majoring, many students do not have the time or resources to undertake a course of study that requires so many extra credits for graduation. Creating hybrid computational-X degrees would allow more undergraduates access to interdisciplinary work.

There are certainly non-trivial difficulties in producing computational-X programs at the undergraduate level. One set of difficulties comes from inside the field itself. In order to make room for undergraduates to study concepts from other disciplines, the course load in computer science must be reduced. Some advanced topics, such as theoretical computer science, compiler design, networking, parallel and distributed computing, and artificial intelligence, may need to be sacrificed to leave room for a significant number of courses in discipline X. Whenever advanced courses are reduced there will be computer science educators who believe that the resulting program will be an inferior version of itself and that such sacrifices should not be made. If all computer science degrees were turned into applied computational-X programs and the advanced topics simply were no longer taught, this would be a justifiable concern. But the goal in producing computational-X degrees at the undergraduate level is to expand the set of people informed by computer science, not to eliminate the opportunities for students to study computer science as a pure field. Just as there are students who choose to study pure mathematics rather than mathematics as applied to another discipline, so should we expect that there would remain students for whom computer science as a pure field would be of interest. Fear that applied computer science would be more attractive than pure computer science is similar to the resistance that computer scientists encounter from the hard sciences when computing courses become options in undergraduate requirements. Computer scientists must have the confidence that what we study is so interesting that there will always be students who wish to join the discipline and find ways to make that a reality.

Difficulties in producing computational-X programs do not only come from disagreements within computer science. In order to understand what courses in discipline X should be added to a computational-X undergraduate program, computer scientists must collaborate in non-trivial ways with educators from those disciplines. In order to be successful, computational-X programs must be designed in a way that makes sense for both disciplines and not be perceived as a grab for students on the part of computer scientists. This means that the people involved in the design of the program should come from both computer science and discipline X, and that both sets of educators should have an equal role in determining the final curriculum. This is a challenging situation since connections between the computer science community and educators from other disciplines are not commonplace. These connections cannot be forced by computer scientists since to do so is intrusive and generally not well received. Finding ways to bridge the divide between computer science educators and educators in other disciplines is by far the most difficult challenge of this idea.

The potential in producing students educated in computational-X is high. Students in other disciplines who also know significant amounts about computer science may be able to ask new and interesting questions about their discipline that people trained in only one field would not be able to see. They may also be able to introduce new problems into computer science that will take the field into directions that would otherwise remain untapped. There is a sort of vision that someone with an equal foot in two disciplines has that is simply unimaginable by those without that training. Ensuring that those individuals can be found in all sectors of society, and not just in academia, is a contribution worth the effort it will require.

Responses to Questions: Kelvin Sung (University of Washington)

CRA Computing Education Workshop:
Pre-Workshop Brain Dump

Traditional **Computer Science (CS) Education** focuses on the disciplinary knowledge. The community has kept up-to-date with innovation and advancements, and been remarkably successful in maintaining the technical pertinence of our curriculum and been moderately effective in educating software professionals.[5] With the ever-increasing integration of technology in all aspects of people's daily life, moving forward, efforts associated with **Computing Education** must demonstrate and establish the wide-ranging relevancy of our discipline and impart the essence of computing knowledge to the general public. In the immediate term, we should focus on the diversification of our ***curriculum*** and on the ***participant***.

The three posted prompt questions can be conceived as approaches to assess the curricula and participants of computing education and to identify opportunities for advancing the goals of establishing relevancy and engaging the public.

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?

Much like biochemistry, depending on the significance of overlaps, new disciplines and/or majors/minors should be established in areas where computing overlaps, e.g., computational science/media, or digital humanities. I believe the key is in the support for effective collaborations:

1. **Foster favorable environments**: Establish institutional infrastructure, resources, and reward systems. For example, there should be administrative support for creating new majors, minors, or refining existing degree requirements to integrate courses from other relevant disciplines. Resources can be provided in the form of funding, and rewards in the form of public awards and recognitions.

2. **Integrate disciplinary knowledge**: Refine and/or create new meaningful curricula. Effective integration of multiple disciplines will require faculty from those disciplines working together in analyzing and re-conceptualizing exiting knowledge. Support for dedicated time of faculty working together is key.

3. **Attract and sustain diverse participants**: Outreach to junior high school with training for the teachers, design multiple pathways to support the diverse student backgrounds and needs (e.g., learning styles and preparedness). It is also important to educate faculty about gender and culture-sensitive instruction.

Many existing B.A. in CS degrees already require an academic minor from another discipline. These curricula can serve as ideal platforms for initial investigations. CS professional organizations like CRA, ACM, or IEEE Computers can play key roles in advocating, garnering resources, educating institution administrations, and establishing infrastructures (e.g., relevant committees and task forces) to enable and support the above efforts.

[5] The lack of participant diversity is one of the main challenges. Although this is related to the *diversification of the participant* discussion in the following, dedicated efforts in understanding and addressing this issue must continue.

I believe the traditional CS education as we know it today will and should continue. Modern society will continue to need core algorithm development, operating system upgrades, etc., and it is absolutely crucial that we continue to assess and address the participant diversity issue in CS education.

2. What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?

"Computing education" is much broader than "enrollments in CS majors." In addition to the new computing related interdisciplinary studies discussed above, it is also about teaching the fundamentals of computing to the masses. This is similar to math education where, in addition to nurturing future mathematicians, it is also about teaching the fundamentals of arithmetic, algebra, and so on to the general public.

The increasing prominence and integration of technology in our everyday life means in the not too distant future, similar to mathematics and English, basic computational proficiency would become part of high school graduation and college entrance requirements. Moving forward, I believe we should focus on:

- **K-12 integration**: "Educating the masses" means identifying and defining the essence of our disciplinary knowledge, assessing the educational opportunities, designing materials and methodologies for deployment to classrooms, and working with the K-12 system for proper integration.

- **Higher ed Integration and service courses**: IT properly support computational competency as a college entrance requirement, we must establish the proper support infrastructure including template placement exams, remedial classes, etc.

  In addition, as part of the investigation into new computing-related interdisciplinary studies, new service and elective classes should be identified and implemented.

While some of the above ideas are long-term or even aspirational, currently there are numerous ongoing outreach efforts in bringing computation-related concepts to all levels of K-12 education. Together, these efforts accumulate frontline knowledge that will support subsequent future systematic integrations. For example, prior efforts and knowledge gathered allowed the implementation of CS principle initiatives and the articulation of the NSF CE21 program.

A study should be carried out to 1) survey all existing outreach efforts, and 2) derive a taxonomy framework to classify the different types of outreach, technologies involved, targeted audiences, contents delivered, etc. These results will allow the identification of overlaps in existing efforts for fostering collaborations and the articulation of underserved areas for additional efforts.

In parallel, it is important for us to continue with the process of building consensus on what are the "essence of our disciplinary knowledge"—what are our equivalents of "arithmetic" and "algebra"?

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

The *direct applicability* of core CS knowledge makes us unique from math and physics. Fresh graduates of CS majors can apply their knowledge and skills directly from course work and accomplish concrete contributions. This *engineering* characteristic is intrinsic to our discipline and will remain.

I expect the number of CS majors to remain significant because of the vast needs and potentials for continuous innovation and improvements of technology integration with all aspects of our society. These efforts will always rely on the core CS disciplinary knowledge and I believe the needs for core CS professionals will continue to be strong. The new computing-related interdisciplinary studies will enhance the presence of CS because many of the new faculty will be jointly appointed and there will be new and interesting sets of elective classes for students. The rich varieties will positively feedback into the continual vitality of CS as a major in higher education.

Responses to Questions: Kathleen Tamerlano (Cuyahoga Community College)

As we begin to think about computing education 5-10 years from now, there are a number of high level ideas/questions that arise:

1. What is the role of CS in truly interdisciplinary efforts versus multidisciplinary efforts? As we move more toward the former, and away from the latter, what are the implications, challenges, and opportunities for CS and computing education?

As more disciplines have a true computing component, it becomes relevant to combine majors, such as computing in medicine, computing in applied math, or computing in finance, where the impact is so strong that the combination adds value to future careers.

This also supports an emphasis on production tools over tools designed purely for the purpose of learning. When students enter the market with industry and production tools, it gives them an edge over their theoretical counterparts and better prepares them for internships. Tools can be identified for a discipline and the integration of these tools provides for a stronger interdisciplinary relationship and connection to careers. From an object-oriented standpoint, it really doesn't matter what tools are utilized to teach objects. It's all the same, and if we focus on the concepts, the tools become superficial. Our students need to realize this as well so they can easily move between fields and adapt to the changing world of CS.

Changing the emphasis to interdisciplinary efforts presents many challenges, such as designing degrees or certificates that will live through at least a few years, applying examples to industry-specific environments while maintaining academic intent, identifying areas of growth with true job opportunities, and redefining these defined areas as times continue to change.

The common denominator is computational thinking where abstraction can pertain to more modern implementations of CS. Interdisciplinary and traditional CS students will continue to arrive with varied backgrounds, bringing different levels of expertise to their first course. Multi-option labs in a breadth of coverage approach can be successful in leveling the playing field while challenging more experienced students and equally preparing students for upper level courses.

2. What happens if we stop always thinking about and worrying about CS recruitment and retention? What happens if we stop worrying about the number of majors and focus, instead, on exposure, broad outreach, butts in chairs across various levels of courses?

When we focus on exposure and broad outreach, we typically have industry partners at the table, as well as academic interests. The influence of industry (tempered with academic interpretation) provides a solid base of interesting components to the computing curriculum that leads to more interdisciplinary majors and greater success. Instead of recruiting for the same foundation majors, providing evidence-based (defined by the job market) degree and certificate opportunities with real-world components in the curriculum may cause a larger number of students to become passionate about their major.

Adding experiential learning to the mix will continue to be a key component in elevating interest and understanding of the various CS opportunities. Understanding can also be clarified by taking a breadth approach to the first experience CS course. If students make a mobile application, Web page, or similar achievement in their first course, they can start to build their computing "toolkit" and start the process of identifying their passion and better understand all of the options available to them.

Providing a variety of hands-on experiences in the first course coupled with career exploration can assist in retaining the interested students. If we did a good job at retention, the numbers would take care of themselves and word of mouth would continue to spread.. Exposure and outreach can contribute to redefining CS and elevating interest, which would be a welcome replacement for recruitment.

Evidence shows students are more engaged when responding to issues directly related to their environment. Identifying such issues and addressing them with experiential and hands-on opportunities can create and spread passion for CS in both interdisciplinary and traditional CS frameworks.

3. What happens (or should happen or should be avoided) as we mature as a field? If we look at math and physics, we see that they have huge service roles, but math still has a solid relatively large number of majors while physics is relatively small. What road do we envision for CS?

With the job market changing to a more applied CS, we need to change as well. In the past many CS majors were working on operating systems and lower level languages; today many of our internships are in .NET, SQL, Java, mobile and Web development. It seems that CS can avoid the label of a "mature field" by adapting to the trends of today's workforce. Such jobs suggest applied CS, just as math has expanded to applied majors as well. From a computing standpoint it may not be as important to focus on gates and circuits as on a broad range of computing applications to best interest and prepare students for future careers while weaving foundation concepts through the curriculum. Once a foundation of interest is formulated, students may better understand the rationale behind upper level CS courses and provide a higher degree of success and retention.

Responses to Questions: Blair Taylor (Towson University)

Major issues in CS and its future:

- *Universities are increasingly being held accountable for employability of its graduates.*

  This means that CS education will have to incorporate more technical skills and internship opportunities. This is a challenge as CS curriculum has typically been largely theoretical with the intention of providing students with foundational knowledge upon which to build and develop technical skills. This emphasis on theory has always been prevalent in CS education.

  CS faces the challenge of preparing graduates in ever-evolving fields. How do we properly prepare students to solve problems and learn the appropriate technical skills to be employable? Increasingly, students and their parents (who are paying hefty tuitions) are demanding marketable skills.

  Industry involvement: One way to address this issue would be to involve industry to keep our students up to date. At our own institution we have used monthly symposiums to keep students (and faculty) up to date on cybersecurity issues. We have also included topics on job preparation, resume writing, etc. While industry partnerships are extremely valuable, it is always difficult to discover which companies are willing to work with us and to determine a point of contact. A database of industries willing to work with universities and what skill sets, projects, etc., they are looking for would be useful.

  Required internship: The obvious challenge with this is there is little room in the crowded CS curriculum. Another problem is the large number of international students in CS. Additionally, this may reduce the number of paid internships as companies opt to award credit instead of pay for student jobs.

  In lieu of required internships, it would behoove us to develop more creative ways to include working experiences into the CS education. More capstone course projects may be a way to address the challenges mentioned and still expose students to real-life problems and provide them with valuable experiences. Following the model of engineering, CS curriculum should certainly place more emphasis on group projects and summative work.

- *Respond to the different styles of learning for Net Generation, Gen XYZ, iGeneration, Generation @.*

  All majors face the challenge of dealing with a new generation of students that are digitally native and highly connected. However, unlike other majors, CS can leverage students' lifelong use of communication and media technology in our teaching practice. My observation is that CS education is not doing the best job of embracing the new technologies into our curriculum and teaching methodologies. Examples would include some use of the flipped classroom (with moderation), incorporating social media in the class (i.e., Facebook and Twitter), cloud storage, etc. Additionally, some of these products are customized for educational usage.

Another issue is tools for faculty in learning environments, such as Blackboard, have not kept pace with the needs of CS faculty to grade projects and assignments that include Web pages, programming projects, database projects, virtual machines, etc., making it very onerous to assign and evaluate atypical assignments.

- *Cybersecurity is a huge challenge across majors and not addressed sufficiently within the CS major itself.*

  I have much to say about this, and I address some of it below in the section regarding interdisciplinary efforts. But the cybersecurity crisis is representative of the challenges faced by CS education and we can leverage the high interest in this field to our advantage, i.e., recruitment, interdisciplinary, industry involvement, etc.

- *Online learning needs to be seen as a benefit, not a detriment.*

  The lessons we learn and the time that we take creating materials for virtual learning can benefit our traditional in-class students. The challenges are tremendous here, the time and resources necessary for creating quality online learning materials are immense. We, as a community, need to pool resources and share technology and tools for providing quality instruction across topics.

- *Interdisciplinary, CS is uniquely positioned as it relates to other disciplines.*

  In terms of developing minors, a possible solution is to develop interdisciplinary minors that are tailored to specific CS topics. A few examples: a cybersecurity minor related to health, an IA minor specific to business and finance, a minor in data mining related to marketing, etc. Developing an interdisciplinary topic-specific minor has its own challenges; it requires a large amount of time and resources, and more funding opportunities for developing such minors would certainly help.

  Additionally, assessing the value in terms of employability is important; here, industry support and participation is important.

- *Computer Literacy courses: What skills are necessary for all undergraduates and are they required? Also, for K-12?*

  Computer literacy as a general education requirement: What should a computer literacy course look like or what computing skills should all undergraduates (non-majors) acquire as part of their general education? (This will also tie into the future of interdisciplinary efforts.)

  At my institution, we have fought to keep computing courses in the general education category. Faculty across other disciplines fail to recognize the importance of a basic computing course or feel that students can learn computing skills in a writing or research class. I strongly believe that we are doing a disservice to undergraduates if we do not require some level of computer literacy skills. This also applies at the high school level. Where does computing fit in? Should there be a CS principles for all? One of the advantages of enforcing a computer literacy requirement and/or CS principles course is increasing interest among undergraduates who may not have a considered pursuing a CS degree. This also applies to K-12 and particularly to high schools where computing curriculum is not adequately addressed, nor sufficiently defined, and the process for systemic change is daunting.

Test-out option: To address the assertion that more students come into college with computer skills, any computer literacy course should have a test-out option.

- *Challenge: Diversify the pool of CS undergraduates*

  The lack of diversity in CS is a crucial problem that is not improving. We need to work together to address the factors that contribute to this. For example, the identity crisis in CS; younger students do not recognize the role of computer science. By increasing computing offerings at K-12 and as general education courses we can help address this issue. More combined efforts with industry can also help here.

- *Different models of schools*

  Computer science started off in math or business. Another example of the identity crisis in CS is where it fits as a major, often straddling the schools of engineering or math and science. Perhaps a model that includes a School of Computing would allow CS the flexibility to offer more degree options, such as an applied computer science degree and more minors, such as the interdisciplinary options discussed above. This also might address some faculty development issues including:

    - Special interests of CS are not well addressed by most institutions. All faculty are evaluated in the same model, resources are allocated in a similar fashion, etc.
    - Encourage and recognize faculty pedagogy and best practices in teaching
    - People of practice to teach courses, more clinical faculty

  This also brings up the emergence of IT as a major and the questions accompanying it. Where does this major fit in and what role does CS play in the creation and support of IT curriculum?

- *Other challenges, issues, and trends that didn't fit anywhere:*
    - Accreditation and certification (i.e., CAE) challenges
    - Re-assessment. What should be core theory for undergraduates looking at some of the traditional CS courses such as AI, compiler design etc.? Assembly language has largely been assimilated into an architecture course. Can we do more of that? Network and network security, for example.
    - Two year schools, flexible