FUTURE DIRECTIONS IN COMPUTING EDUCATION SUMMIT PART ONE:

IMPORTANT COMPUTING EDUCATION RESEARCH QUESTIONS

A report of the January 8-9, 2014 summit held in Orlando, FL.

CS-TR-14-0108-SC, Stanford University, 2014

ACKNOWLEDGEMENTS

The summit and this report are supported by NSF-1332686. Any opinions, findings, conclusions, or recommendations expressed in this Report are those of the authors and do not necessarily reflect the views of the authors' institutions or the NSF.

Special thanks to Kathy Menchaca, who helped to organize and run this meeting.

Thanks to Engin Bumbacher, Shuchi Grover, Chris Piech, and Marcelo Worsley for providing their notes to help in preparing this report. The report was prepared by Steve Cooper, with significant input from the summit attendees.

To cite this report:
Cooper, S., Bookey, L., & Gruenbaum, P., 2014. Future Directions in Computing Education Summit Part One: Important Computing Education Research Questions, Orlando, FL, January 8-9, 2014. Technical Report CS-TR-14-0108-SC, Stanford University, 2014.

Important Computing Education Research Questions

Executive Summary

The National Science Foundation (NSF) funded a summit in January of 2014 where researchers gathered to discuss future research directions for Computer Science (CS) education. The participants addressed the question: “What do you consider an important research question, or series of related questions, in CS education research?” This report summarizes the discussions from that summit.

- The most significant question raised in the summit was “How do we involve *everyone* in computing?” How do we provide computing education for *all*? Our “everyone” includes females, underrepresented minorities, low-income students in community colleges, as well as students with disabilities. When we talk about computing in compulsory education ages, we also have to consider how we provide computing education to people who may not be interested in computing or in STEM. We also have to think about computing for a broad range of abilities.
- The challenge of preparing enough teachers to meet our school-age needs in computing education is so large that we might consider an alternative approach. The summit participants explored the possibility of introducing computing through existing mathematics and science classes, using STEM-prepared teachers to learn enough computer science to teach introductory units. A growing body of research shows how programming activities can enhance learning in science and mathematics.
- We know too little about *how* learning occurs in computer science. Foundational questions to be answered includes "How do students learn to program and what does that development look like?", "What are successful and unsuccessful mental models of challenging computing concepts?", "How do we support successful transfer from beginner programming environments to real-world ones?", and "What are common challenges in conceptual understanding in computing course?"
- Computing and STEM share a symbiotic relationship. STEM can enrich computational learning while also providing valuable opportunities to embed CT in established and accessible (as well as required) STEM courses. The CT-STEM research agenda would thus need to seek answers to several questions, including, “What CT skills are most important for STEM?”, “How can these skills best be taught through curricular units in STEM coursework?”, “What STEM content can be reinforced naturally through CT?”, and “How can we get STEM teachers to adopt and practice computational methods and approaches in their curricula?” These questions are also worth asking for particular STEM disciplines. What do Engineering faculty (for example) need to teach about computing, and how can computing help with engineering education?
- Most STEM discipline-based education research (DBER) areas have been around longer than computing education research. We need to learn lessons from these other DBER efforts to grow computing education and avoid the mistakes that others have faced. We have to identify the

barriers to greater access and success in computing education in order to address those. For example, do we have requirements for mathematics that are greater than are necessary for success in computing?

- We know too little about how to assess learning computer science. We need to improve assessment if we're going to answer the foundational questions about how knowledge in computing develops. Summit participants are exploring the possibility of using "Big Data" to improve our understanding of computing knowledge development.
- The greatest need in providing computing education in schools is to provide more teachers. Many of the participants are exploring how to provide professional development and how to motivate teachers to learn computing. But some of the participants are exploring how we can help non-CS teachers to teach CS, the role of information education options (non-profit organizations, start-up companies, MOOCS, and on-line video sites).
- There has been a recent significant interest in learning analytics, tools, and grading, seen especially with the recent creation of the Learning@Scale conference. While Learning@Scale is not limited to studying CS, that seems to be the focus of early research. There is an opportunity to synergistically incorporate researchers and research from this growing Learning@Scale community into the more traditional computing education research community.
- Beyond getting students and teachers interested in computing, we need to understand how to improve the quality of graduates. We need to understand how to prepare computing graduates to use the most effective development processes, including team behaviors and parallel programming abilities. We need to prepare all STEM students to have the kind of fluency that they need with computing.

Introduction

On January 8 and 9, 2014, researchers primarily from the United States gathered at a summit to discuss future research directions for CS education. The summit was funded by the NSF and was organized by Steve Cooper of Stanford University, and by Mark Guzdial of the Georgia Institute of Technology and by Beth Simon of the University of California San Diego.

The challenge posed to the summit was to identify the key research questions in computing education. Industry in the United States is desperately short of highly-qualified applicants for computing jobs. STEM students today will likely be interacting with programming code at some point in their career, even if they are not professional software developers. There is increasing interest in providing computing education earlier, in high school, or even middle or elementary school. We know little about how to achieve all of those goals. What might we do in computing education research to meet our needs in industry, in STEM careers, and in school age curriculum?

Process

Each applicant for the summit was required to write a white paper about research directions that interested them. From the more than 100 white papers submitted, the summit organizers (in consultation with NSF) identified the white papers that were to be accepted. The lead author from each accepted white paper was invited to identify one representative from the author list to attend the summit. These accepted white papers (available from: http://purl.stanford.edu/mn485tg1952) were made available to each summit attendee. A few attendees gave presentations, and in addition, there were presentations from people outside of the CS education field that covered education research from other disciplines (physics and engineering education).

After each presentation, there was time for discussion. There were also three breakout sessions where attendees discussed specific topics. These topics were generally chosen by the organizer, but in some cases, attendees decided to create their own topics.

This report primarily summarizes the discussions and breakout sessions, although occasionally there is information about the presentations in order to provide context.

White Papers

Professors from a variety of universities and colleges with strong computer science departments, as well as past recipients of NSF grants, including recipients of CCLI/TUES awards and attendees at CE21, were invited to submit a white paper to be reviewed by committee. Additionally, e-mails were sent to various listservs where we thought there might be faculty interested in attending the summit. Applicants were instructed to address the following in their white paper submission:

> "What do you consider an important research question, or series of related questions, in CS education research? These questions should result in opportunities for a fruitful research program that will contribute evidence-based findings to the body of knowledge on teaching and learning of Computer Science within diverse student and teacher populations. You need to provide justification about the impact this research will have, including how the US would benefit from undertaking this research direction, what might result from this research program over the next five years, and how it will alter or impact what has been done in this area to date. The results of the research should inform the computing community's understanding of effective teaching and learning in computing for all students, including those groups underrepresented in computing."

The accepted white papers (available at http://purl.stanford.edu/mn485tg1952) asked research questions primarily in five areas: broadening participation, computing in K-12 (primary and secondary) school, computing in Science, Technology, Engineering and Mathematics (STEM) education, students and learning issues, and tools. A few papers were written at a meta-level, looking at the direction of computing education research (CER) and its impact, and looking more broadly at challenges facing the teaching of computing at all levels.

Broadening Participation

Papers made a broad call for "computing for all." Several disadvantaged groups were identified and their cause represented in the papers. These included females, underrepresented minorities, low-income students in community colleges, as well as students with disabilities.

The need for incorporating an intersectional perspective into gender and computing studies was urged since perspectives and experiences are shaped by the constant navigation of multiple shifting identities of gender, race, class, religion, and ability, among others. As an immediate and concrete step, researchers were urged to disaggregate empirical data in CER based in race, gender, socio-economic status, etc. Studies at the intersection of race and gender, in particular, were urged, to better understand why certain groups choose to enter and persist in computing while others don't. To this end, the need for qualitative research methodologies (such as interviews and case studies) was underscored, so the richness of personal narratives could inform action. Papers pointed to several ideas, including:

- Studying ways to better understand how conflicting cultural values and self-belief that impact motivation, and tackling these through design practices and tools that foster meaningful learning.

- Improving computational tools and programming environments to be more accessible to the hearing and visually disabled, and increasing collaboration with AccessComputing .

- Building on the exploring computer science (ECS) course (available from http://www.exploringcs.org/), computer science principles (CSP) framework (available from http://apcsprinciples.org/) and related work to understand better how teachers can be prepared to make "classroom magic" happen.

- Exploring the humanities (including creative writing, theatre, and media production) as vehicles to carry Computational Thinking (CT) agendas (Wing).

Computing in K-12

Since computing education falls in the realm of discipline-based education research, it was not surprising several papers argued for learning research in computing education. Foundational questions such as "How do students learn to program and what does that development look like?", "What are successful and unsuccessful mental models of challenging computing concepts?", "How do we support successful transfer from beginner programming environments to real-world ones?", and "What are common challenges in conceptual understanding in computing course?" were listed among the many key questions the learning sciences can help address.

Papers advocated for giving due importance to the social dimensions of computing education, suggesting strategies for adopting approaches that encourage participation. Some advocated for the use of "model-based" thinking and practice to foster and promote CT. Some authors argued for the need for supporting the professional growth of computing teachers as well as building a robust community of computing teachers. How, for example, will the US meet the CS10K challenge (Cuny) by 2016? Authors also underscored the need for the comprehensive development of Pedagogical Content Knowledge for teachers who teach computing, as well as ideas for just-in-time teacher professional development. Suggestions were made to establish best practice and define teacher roles in classrooms where freely available online curricula were being used, to develop robust assessments, and to study various introductory programming environments to better understand their optimal use in K-12 classrooms to meeting computing learning objectives.

Computing in STEM Education
Computing and STEM share a symbiotic relationship. STEM can enrich computational learning while also providing valuable opportunities to embed CT in established and accessible (as well as required) STEM courses. We still struggle with providing adequate computing experiences to all students as part of K-12 education. CT through STEM can also address the issue of teacher shortages, as training STEM teachers to use CT within the context of content areas they are familiar with may be more practical and amenable to implementation at scale. The CT-STEM research agenda would thus need to seek answers to several questions, including, “What CT skills are most important for STEM?”, “How can these skills best be taught through curricular units in STEM coursework?”, and “How can we get STEM teachers to adopt and practice computational methods and approaches in their curricula?”

Modeling and simulation were proposed as concrete areas where computing and STEM can benefit the learning of both the STEM content and the development of computational skills. There were also calls to dovetail efforts with the Engineering Education community.

Students/Learning
There were several calls for data and evidence-driven analyses of student learning as part of adding rigor to drive practice in CER, including the use of big data to develop scalable methods for automated assessments (along the lines of emergent forms of assessment in massive open online courses (MOOCs). The oft-heard refrain in several papers was to leverage big data to understand better the nature of learning programming and computational problem solving, the problems learners encounter, and the pedagogies and strategies that can be used to address them. Papers urged the use of evidence-centered design and data mining for the study of both cognitive and non-cognitive aspects of CT.

Researchers advocated for the definition and validation of learning progressions for K-12 computing. Some researchers called for studying how people “become computer scientists” and to investigate the broader meanings of the computing discipline. The use of design-based research and design-based implementation research was encouraged to support research and assessment in naturalistic settings. Other unique ideas presented included the teaching of parallel and distributed computing, spatial skills, as well as creative thinking in computational problem solving.

Tools
Several researchers had built and developed curricula around tools for teaching and learning computing. For example, Dann has worked with Alice, Rodger has worked with JFlap, Fisler has worked with Bootstrap, and Repenning has worked with AgentSheets. The conference organizers ultimately chose to fold these papers into other topic areas, though in hindsight, we should probably have kept a focus area on tools, as they have historically been a significant part of computing education research and practice.

The Summit

Computer Science for All
CS education has two objectives: creating CS majors and teaching everyone some level of how CS works (“CS for all”). There may be different strategies for these two goals, but they are not incompatible. An

analogy to consider is that Brazil has such good soccer players because everyone in Brazil plays soccer. By providing CS for all, we can increase the number of CS majors, and we can more easily find high-performance software developers. In addition, if we can get everyone to think "like a computer scientist", we will increase the success of computing being used in other fields. The success of code.org's Hour of Code demonstrates unprecedented attention to coding, and we should capitalize on this.
A few thoughts about research: research questions should consider how they will or will not address "CS for all", understanding that not all research can address everything. By default, research questions should be assumed to be as inclusive as possible.

This section describes the concepts of fluency, literacy, and expertise when thinking about CS for all. It also discusses the question, "Can anyone learn to program?", barriers to CS for all, issues in K-12 CS education, alternatives to CS teachers, and broadening participation. Each section begins with key questions used to prompt discussion.

Fluency, Literacy, and Expertise

How does creative expression compare to puzzle-solving in terms of effectiveness in developing computing concepts? How can we measure progress with fluency? What are the learning outcomes in the context of "CS for all" at the middle school, high school, and university level (majors and non-majors)? What kinds of context and activities make computing concepts more relevant to non-majors?

Mitchell Resnick of MIT discussed computing "fluency" through the Scratch programming language that his team has developed. He compared it to how people learn to write – not everyone will do it professionally, but it's still very useful for everyone to learn. Fluency is creative – the paint editor was a critical part of Scratch. He argued that this creativity and expression was very important, pointing out that you cannot learn to write by solving non-creative problems such as crossword puzzles.

There is a tension between problem-solving and expression. However, it's important to note that problem-solving is required to attain expression and design; the design process is a series of mini-hypotheses. In the school system, problem-solving is prioritized, whereas in informal settings, expression is prioritized. There is a place for puzzle-solving, but some argue that it is currently over-prioritized; prioritizing expression opens it up to a broader audience.

Literacy, compared to fluency, is typically considered less about expression. Literacy often means familiarity with major works and reasoning. However, Sylvia Scribner provides a useful definition of literacy with three types: functional (doing things), personal (expressing oneself), and political (acting as citizens) [Scribner]. We teach reading and writing literacy because we want students to be good citizens that participate in society; perhaps computing is similar in this aspect. We've had mixed success in teaching reading/writing literacy by identifying and teaching the skills – assessments are very positive, but few people read for pleasure.

How can we tell if we are measuring progress with fluency? Unlike physics, we don't have an agreed-upon set of concepts that we want the students to learn. The goals for Scratch were diversity and complexity in the projects created. Cleary, more than one type of assessment is needed to measure fluency. Resnick argued that if adults can assess each other without tools like multiple choice tests, why do we need it for children?

We need to decide on learning outcomes in the context of “CS for all” at the middle school, high school, and university level, where at the university level, we are talking about non-majors. In addition, we need to figure out what kinds of context and activities make computing concepts more relevant to non-majors.

Can Anyone Learn to Program?

How do intellectual ability, cultural perspectives and temperamental factors affect the ability to learn to program? What level of proficiency do teachers need? Does everyone need to know how to program? Are there computer languages other than those used by professionals that work better for beginning programmers?

In addressing this question, it’s important to note that there is no clear definition of what it means for someone to know how to program. It will be different for higher education compared to K-12, and even for higher education, it will be different for CS majors and non-majors. When discussing innate abilities, one needs to look at intellectual ability, cultural perspectives, and temperamental factors (such as persistence and attention to detail).

If you ask, “Can everyone learn to write?”, then the answer is clearly “yes,” even if everyone cannot learn to be a great novelist. It may be similar for programming, but, like writing, we should strive to get beyond the basics. This is difficult to do if the teacher doesn’t have proficiency, and teachers with programming proficiency will often leave for industry, which pays more.

Does everyone need to know how to program? Industry sometimes pushes back against a desire to teach everyone, claiming that they don’t want to deal with bad code, but this may be an elitist attitude. Also, it’s important to note that the computer languages used by professional software developers may not be appropriate for CS for all.

If the community decides that everyone can program, then it will support the exceptional programmers; if it does not, then it must work to find the exceptional programmers in order to support them.

Barriers

What barriers prevent students from pursuing CS? Are there unnecessary math requirements? How can we get policy makers and parents on board? How can we get around CS’s image problem? How can we disseminate research findings? How can we reach “invisible” students, such as those with disabilities? Would requiring CS for all become a barrier to certain students, such as those in special education programs?

There are barriers to all students learning CS. High math requirements such as calculus may not be necessary. We need to figure out how to get buy-in from policy makers and parents. CS has an image problem, where sometimes face-saving techniques seem necessary to get students into the field. When we mention “for all”, we have to think about the family and community context in addition to the students.

Other barriers include lack of support at the institutional level, lack of research for applied work, challenges to disseminating findings, the slow pace of research, being clear about what “CS for all” is, and trying not to perpetuate stereotypes.

Although schools reach most young people, getting CS to everyone will require more so that “invisible students”, such as those with disabilities, will be reached. For example, Scratch does not work for everyone, such as people with visual and physical disabilities, as well as people who have interests other than creating games and animations.

In some special cases, we don’t want learning CS to become a barrier; for example, where special education students are prevented from obtaining high school degrees because they are unable to code.

K-12

What are the learning objectives and learning progressions for each level in K-12? Can we use automated assessment to assist teachers, helping new teachers grade more consistently? Is getting CS into standardized tests worthwhile?

K-12 literacy will be different than higher education. K-12 allows for exploration of multiple approaches, whereas in higher education, CS1 is being taught for people who want to become computer scientists, and it needs to be changed to be accessible to everyone.

We need to define learning objectives for each level in K-12. It would help to understand the most common misconceptions that students acquire while creating mental models. Without consensus on learning levels, assessment is very difficult, especially for programs like Khan Academy and code.org. What is needed to understand the learning progressions, such as the best progression of topics, the appropriate background of next topic, and how to think about understanding learning progressions empirically? What are effective pedagogies and tools for introducing concurrency concepts to younger students?

Grading is important in K-12. Can we use automated assessment to assist teachers, helping new teachers grade more consistently? How do you create a rubric, and can automation help assess this type of situation? Some challenges arise, such as how do you grade an assignment of the form “create a fun game”?

How critical is getting CS into standardized tests? Testing is controversial, with advantages and disadvantages. For example, having standardized tests affects how a subject is taught.

Alternatives to CS Teachers

What are the most important concepts and skills to communicate to non-CS teachers who are teaching CS? What is the role of informal education options (non-profit organizations, start-up companies, MOOCs, online videos, etc.)? Should computing be taught in other domains, such as art and science rather than through stand-alone CS classes?

It is already starting to happen that non-computer science teachers are teaching computer science material, and not in the same way that computer science teachers teach that content. We need to be supporting and encouraging them. It would be useful to identify a very short list of the most important concepts and skills and communicate these to others.

There are informal educational options outside the schools that will come into play, such as MOOCs, iTunesU, non-profit organizations, and start-up companies. Libraries are reshaping themselves and are interested in CS. What can we learn from these options and organizations? Out-of-school time plays an important role in proficiency, because it's impossible to put in enough hours to become an expert if students only learn during school time. Literature exists on the psychology of out-of-school organizations.

Should computing be taught in other domains, such as art and science rather than stand-alone? By packaging pedagogy and technology together, we can export computing to other disciplines. The problem is that other disciplines must cover a certain agenda of material, and there may not be room to add in computing. Ultimately, we need to be supportive of both stand-alone courses and integration.

Broadening Participation

How important is "loving the machine" to success in CS, and how does this differ for different genders and cultures? How does masculinity play a role in CS culture, and how does this affect both girls and boys? How can we go beyond demographics to look at shifting categories of identities (i.e., intersectionality)? What is the current culture of CS regarding these issues (for example, "loving the machine") and how can it be changed? Do CS majors need to join the CS culture to be successful and have it be one of their "tribes"? How do we design culturally-responsive computing curricula? How do we build pathways from K-12 to higher education (both 4 year and community college) to industry (both software developer and other computing jobs)? What are the motivations and values of underrepresented students?

Making a distinction between "CS for all" and "broadening participation" is difficult because they are so closely related, but whereas "CS for all" is focused on how to get everyone to learn at least some level of CS, "broadening participation" tends to be more focused on women, underrepresented ethnic groups, and people with disabilities, looking at not only getting them a basic level of CS knowledge, but as becoming fully represented in all aspects of computing fields.

Catherine Ashcraft of the National Center for Women and Information Technology presented information on intersectionality and masculinity. (Intersectionality means multiple shifting categories of identity, including gender, race, class, ability, and sexuality; it involves not just demographic categories, but identities and choices.) Clayton Lewis of the University of Colorado Boulder explained how visual programming can be used by people with visual disabilities, demonstrating that visual programs are just spatial databases that can be accessed through interactive audio. Many common educational tools, such as Scratch and Alice, are unusable by visually-impaired and mobility-impaired students. (However, someone pointed out that professional developers often use IDEs that are more accessible.)

Underrepresentation affects many STEM fields besides CS. Physics still falls short when it comes to women and underrepresented ethnic groups, and they don't study this as much as they should. Motivation is being studied, but not applied to diversity.

We need to call into question whether you need to "love the machine" to succeed in CS. Are we setting up certain students to fail if they don't love the machine? Love means different things to different genders and cultures. Changing this perception means not just changing the language, but changing CS culture itself.

Research into masculinity is useful in understanding how to broaden participation to girls. For example, the phenomenon called "Brogramming", where programming is part of male-bonding, has some positive aspects, but it can marginalize women. When creating girls-only classes, sometimes boys ask why there are no boys-only classes. Boys of color also feel like outsiders.

What can we learn from female-dominated industries, such as teachers and nurses? For one, female-dominated jobs are less well-paid and valued, which is why we focus on girls learning skills in STEM. Expanding what's okay for boys to do expands what's okay for everyone to do.

Thinking in terms of demographics has limitations. Everyone's brain is different on the inside, and there's more to people what demographic group they belong to. In addition, different cultures have different biases – for example, in Israel, Orthodox Jewish women are encouraged to become programmers.

Discussions on broadening participation sometimes divided into two categories: how to change the culture of CS education, and how to reach out to broader groups of people. We need to understand the current CS culture in order to figure out what needs to be changed, and identify the values, barriers, and leverage in changing faculty and curricula. For example, requiring calculus hurts broadening participation. Do CS majors need to join the CS culture to be successful and have it be one of their "tribes"?

To reach out to broader groups of people, we need to think about curricula, pathways, and motivation. How do we design culturally-responsive computing curricula? How do we build pathways from K-12 to higher education (both 4 year and community college) to industry (both software developers and other computing jobs)? We need to understand the economics of participation. We also need to understand the motivations and values of underrepresented students. For example, some are motivated by creativity and others by having good jobs.

Satisfying the Demand for CS Students and Teachers

Software companies need CS majors to hire and schools need qualified teachers to teach CS. This section covers student beliefs that may prevent them from taking CS classes, the growing concern of having enough CS teachers to teach students who do want to learn CS, how online learning and automation can assist in satisfying the demand, and how to teach students what they need for a computing profession.

What Are Student Beliefs about CS and How Does It Affect Their Choices?

How can we address beliefs (often misconceptions, but not always) about programmers working by themselves, computer science being too difficult, and students not "fitting into the CS tribe"? How do we *expose students to programmers in industry? How do we increase exposure to diverse programmers? How do we reach geographical areas that don't have strong technology centers?*

Students have several beliefs that can influence their decisions about learning CS. Some of these beliefs persist because they have some validity. These beliefs were discussed:

The Lone Programmer. Working by yourself is not appealing to many students. Students are often not aware that programmers commonly work in teams in industry, although it is worth noting that many programmers do work alone. Exposure to programmers in industry can change misconceptions, and providing more opportunities for undergraduates to work together will give them a sense of what it's like. As an example in another field, medical school interviews have people working in groups, and then evaluating how they work.

Computer Science is too hard for me. Computing seems to be a discipline where ego and insecurity run rampant, sometimes even in the same person. Students are not willing to risk their GPA, so perhaps having credit/no-credit courses may be advantageous. Even if students are doing well in class, they may believe that real programming must be much harder than what they are doing.

Not fitting into the tribe. There is a trend where programming becomes part of young male bonding, called "Brogramming" -- combination of "bro" (slang for brother) and "programming". This can make it harder for women to feel included. In addition to gender issues, there are geographic factors, such as students who grow up in the Bay Area are much more likely to be exposed to computing ideas that those who grow up in Wyoming. Student teachers are a good way to bring in diversity and help with this issue.

In conclusion, it was noted that perception is brittle and often outside our control. However, awareness breeds more awareness, and if entry courses are taught well, that can have a big impact on perceptions.

The Need for CS Faculty

For top tier universities, how do we increase the number of CS teachers to match demand? How do we ensure CS teachers are not so overloaded that they cannot provide quality education and so they leave? How do other disciplines deal with similar situations?

Eric Roberts of Stanford gave a keynote address about top tier universities, where the number of students interested in majoring in computer science is growing rapidly and the demand for teachers will be hard to meet. There was a similar crisis in the 1980s, and it was very difficult. Adding up all of the STEM majors will cover all STEM jobs; however, if you break it down by type of STEM degree and job, there are many more CS jobs than CS majors. Industry is looking for very high performance developers, whose productivity is much more than average; this is why they are so picky in hiring.

When there is not enough faculty, innovation in education suffers. It can be argued that the poor pedagogy was a survival strategy to deal with too many students. Faculty will leave if they cannot provide the quality education that they want to give their students because they can get higher paying, lower stress jobs in industry. One short term solution is to eliminate students, but this not a good long term solution.

Economic departments have similar issues with faculty leaving for industry. In physics, they reduced the number of graduates because physics jobs were so scarce. The telephone industry gave women training to become installers when they were short. The software industry sometimes does this, giving intense training to non-majors.

Online Learning and Automated Assessment

What are the learning impact advantages/disadvantages of different on-line or computer-mediated instructional techniques, such as video lecture, MOOC-style discussion, on-line problem-based learning (PBL), etc.? What are the most effective ways to keep students engaged in online learning? What factors are critical to success in self-paced online learning? How does online professional development work with different types of learning – broad vs. deep learning? Can we build automated tools to give students a grade on a specific assignment and what is the level of feedback? Can we extend automated assessment into interactive learning? How can we use data from automated analyses of code to study learning processes?

Online teaching and automated assessment are two tools that can help bring CS education to more students. The following research questions were raised:

Online learning. What are the learning impact advantages/disadvantages of different on-line or computer-mediated instructional techniques, such as video lecture, MOOC-style discussion, on-line PBL, etc.? What are the most effective ways to keep students engaged in online learning? What factors are critical to success in self-paced online learning? How does online professional development work with different types of learning – broad vs. deep learning?

Automated Assessment. How far can we really push automated assessments? Can we build automated tools to give students a grade on a specific assignment and what is the level of feedback? Can we extend automated assessment into interactive learning? How can we use data from automated analyses of code to study learning processes?

How to Prepare Students for the Professional World

How can teamwork be encouraged through collaborative technology (such as source control), visualization of data, and capstone projects? How can students work across disciplines, and how can they learn the language of other domains? How can use of computer languages be studied in terms of effective learning and effective use in industry?

Higher education can be better at preparing students for a computing career. The following ideas were discussed:

Team work. Team work is critical. Students should know how to use collaborative technology, such as using source control tools to merge and branch projects. Communication is also critical, using both language and effective visualizations of data. Grading currently favors individual work. One potential solution is capstone projects, which should be used early on, and continuously, not just in the last years of undergraduate studies. Capstone projects provide motivation through relevancy, they show the value of CS in other fields, they foster common language and mutual appreciation across disciplines, and they avoid the administrative hurdles of actually creating a new program, but still have the benefits of interdisciplinary interaction, and they provide databases with data from other fields that CS students can work with.

Interdisciplinary Application. CS is blending with other fields such as computational biology, physics, social sciences, etc. There is a "Big Data crisis" where there is so much data generated and not enough understanding of what to do with it. CS people are often used as data crunchers in other disciplines, so it is useful if they know the languages of other domains. People outside of CS should know at least something about the language of CS. However, CS students often don't know their value in the other fields, as there are no collaborations or interactions between the disciplines at universities. Also, CS students rarely have the opportunity to use the skills they learn in other domains, which can often make the work more meaningful to them. Interdisciplinary structure can assist with this, such as joint appointments of CS professors and double major/minor degrees. Collaboration on a faculty level can trickle down to the undergraduate students.

Languages. The educational systems use a large number of different programming languages. There is a proliferation of different languages after introductory classes, many of which are not relevant to industry, and very little evidence about what is effective for learning. We need to evaluate the impact of languages on CS learning, including a reductionist approach where you determine whether individual features that work or not is needed.

Structure, Policy, and Practice

This section covers areas that involve taking a step back and looking at CS education at a more abstract level for both student learning and teacher learning.

Student Learning

How do students go from beyond concrete to abstraction? Are there tools available that can help with the abstraction process? How do you challenge students to make more elegant coding? How does problem-solving and creativity fit in? What are the mental models we expect students to acquire as they learn how to program, such as how to debug, how to design software, and how computers behave? What are the most common misconceptions that students acquire while developing their mental models of how programs operate, and what instructional techniques are most appropriate for correcting these misconceptions?

How do students go from beyond concrete to abstraction? One person's idea of abstract may be another person's idea of concrete. In general, students switch from concrete to abstract constantly, where having latent knowledge allows them to make the abstraction. In general, we tend to start with teaching the concrete and hoping they abstract. One suggested method for teaching abstraction involves waiting until students have a few contexts and see if they will abstract. Then help them find the commonality and lead them through the abstraction.

Students learning Java are forced to abstract earlier because object-oriented programming requires a leap to abstraction. However, students can get bogged down on syntax. Are there tools available that can help with the abstraction process? How do you challenge students to make more elegant coding?

How does problem-solving and creativity fit in? Turkle and Papert suggest that there is no formal divide between the abstract and concrete[S. Turkle and S. Papert].

There are several questions to be answered about mental models. What are the mental models we expect students to acquire as they learn how to program, such as how to debug, how to design software, and how computers behave? What are the most common misconceptions that students acquire while developing their mental models of how programs operate? What instructional techniques are most appropriate for combating and correcting these misconceptions?

Teacher Learning

How do we put information about research results in places where teachers will look and what structures are required for adoption? How do we determine what we want disseminated? How can researches learn what teachers want, and what their thoughts, experiences and concerns are? How can departments provide help for teachers?

Jeanne Century of the University of Chicago covered how STEM K-12 teachers obtain what they need to teach, thinking about resources beyond textbooks, the types of studies needed, and how resources are designed and delivered. There is a unique opportunity for CS community to use lessons learned in STEM, and collaborate and communicate on best practices contents.

Sally Fincher of the University of Kent presented a keynote that discussed how teachers learn and how they change their practice. She talked about the "stuff fallacy," which is the belief that if you have better stuff (best practices, etc.), then teaching will improve. She also provided findings about what teachers read, and how they gather information that they will use at some later point, but not necessarily in an organized fashion.

Teachers who are motivated and knowledgeable are good at finding and curating new information, but many teachers do not know how to do this. Therefore we need a foundation of materials, but it's important to note that dissemination is not the same as adoption. How do we put the information in places where teachers will look and what structures are required for adoption? Teachers will adapt the material, which is good, but we want principled adaptions.

Is there even enough CS education research that we want to get out into practice? How do we determine what we want disseminated? A more bi-directional relationship between research and teaching is needed. Researchers need to know what teachers want, and what their thoughts, experiences and concerns are.

How can departments provide help for teachers? University of Washington has "trouble-shooters", who are practitioners who can help faculty members with issues in the classroom. Any stigma with using one of these troubleshooters is lessened by applying confidentiality.

Incorporating CS into Other STEM Classes

What is the ideal amount of CS learning to incorporate? How can non-CS teachers be motivated to teach CS? Are strategies from computer science applicable and adoptable in STEM courses? How well does learning computational thinking (CT) in STEM courses impact computer science learning and transfer? Do the assessments from CT in STEM carry over into CS courses? What would CS STEM look like at K-8 and 9-12? How can CS be added systematically into STEM courses in ways that are scalable? How would learning progressions work together between CS and STEM? What factors are impacting the lack of diversity in both STEM and CS? How can students see the broader range of connections between STEM and CS? If we compare how students perform in STEM and CS, can we see if the combination enhances performance?

More detailed research questions include: What are the barriers of each state in implementing any CS in STEM? To what extent do teachers understand the concepts of the Next Generation Science Standards STEM requirements, how can we effectively teach in line with NGSS, and can we assess using NGSS metrics? What are the appropriate computer languages to use?

In many schools, especially K-12, CS classes are very limited. By incorporating CS into other STEM classes, CS concepts can be taught in a way that enhances the subject being taught, as well as teaching computational thinking. Common Core testing is being done electronically, which results in more computers in the schools. These computers can be leveraged for more CS in math and science.

The CS learning required to experience the math and science portion cannot be so high that it eclipses the math and science curricula; on the other hand, it cannot be so little that there is no effective CS learning[Basawapatna, A. Repenning, A]. Transfer from one discipline or context to another is difficult, and some bridging is needed [Engle, et al]. We first need to identify structural similarities in the problem-solving used when teaching math and science.

Teachers are motivated to learn CS to make their classes better, but they do not want to be experts. It may be easier to train math and science teachers to teach CS than to provide CS teachers, but they need to be motivated to do so. Some math teachers are frustrated with the level of algebra learning at the middle and high school level, and CS provides exciting alternatives.

In order to incorporate CS effectively, we may need to work with the Next Generation Science Standards (NGSS). We need to understand to what extent teachers understand the concepts of the NGSS STEM requirements, how CS can effectively teach in line with NGSS, and if we can assess using NGSS metrics. A mapping between concepts across standards could be useful.

In middle school, students can use and modify simulations. This can then progress to high school, where computational skills are better developed, and simulations and models can be created from scratch. One potential tool for incorporating CS into STEM classes is the simulation software NetLogo. Teachers can learn how to do a unit of NetLogo with 6-10 hours of professional development.

For higher education, the computer languages used by professional software developers may not be appropriate for other domains. Computer science education researchers need to better understand the needs of STEM workers who do computing in other domains. Even when different computer languages are being used for different disciplines, we need to determine the general core everybody should know.

In theory, CS can be applied to areas outside of math and science, such as social science, demonstrating how economic inequality occurs. Unfortunately, there is not momentum in these areas.

Research is needed to answer questions such as:

- Are strategies from computer science applicable and adoptable in STEM courses?
- How well does learning computational thinking (CT) in STEM courses impact computer science learning and transfer? Do the assessments from CT in STEM carry over into CS courses?
- What are the barriers of each state in implementing any CS in STEM?
- How can CS be added systematically into STEM courses in ways that are scalable?
- What would CS STEM look like at K-8 and K-12?
- How would learning progressions work together between CS and STEM?
- What factors are impacting the lack of diversity in both STEM and CS?
- How can students see the broader range of connections between STEM and CS?

If we compare how students perform in STEM and CS, can we see if the combination enhances performance?

Research Methodologies

How can we educate the reviewer population about research methodologies, including grant, conference, and journal reviewers? What are good ways of capturing contexts in CS educational research? How do you represent the contexts in which the research was conducted? How can we apply methods from the social and learning sciences that are not typically part of CS research, such as case studies, interview analyses, narratives, critical ethnography, descriptive statistics, market research techniques, and design-based research? How can we use CS to apply methods not typically used in social sciences, such as educational data mining, big data analysis, machine learning, AI techniques, and natural language processing? Often funding institutions favor research, but do not fund the maintenance of the tools for implementation because it is not new work – what organizational systems and what supports need to be put in place in order for implementation tool creators to maintain their tools? How can data best be shared?

Will Doane from University at Albany discussed computing education research, including CE21 projects evaluation, the need to develop comparable research materials, embrace multiple research methods, and be aware of other research traditions and compare results. Three critical elements are communication, documentation, and transparency.

The reviewer population needs to be educated about research methodologies, including grant, conference, and journal reviewers. One way to achieve this is to propose a special issue of TOCE or Computer Science Education on the subject. We can educate students using existing research texts, such as texts on research design [Creswell].

Post-hoc statistical methods have as much quantitative rigor as do randomized control trials. Anecdotal isn't the opposite of quantitative; it's the opposite of rigor. And, rigor is achievable in both quantitative and qualitative research. There are a plurality of both empirical/research methods and analysis methods. Large and small studies are both effective, as long as the appropriate techniques are used.

What are good ways of capturing contexts in CS educational research? How do you represent contexts which the research was conducted?

Because computer science is a cross-disciplinary effort, we need to have awareness, appreciation, and acceptance of cross-disciplinary methods. We can learn from a range of methods from the social and learning sciences that are not typically part of CS research, such as case studies, interview analysis, narratives, critical ethnography, descriptive statistics, market research techniques, design-based research. In addition, we can use CS to apply methods not typically used in social sciences such as machine learning and AI techniques. The field of learning analytics needs to be grown through educational data mining, natural language processing, machine learning, and big data analysis.

There are people from outside that could help CS education, but it requires coordination and communication. There are opportunities to put structures in place to enable accumulation of knowledge around instructional material, and increase the return on investment. We want to see what the identification of the key issues are in going forward, and maximize and leverage all the knowledge we have, and see how to move on from there.

How do we enable sharing information and bring people together from studies that are not NSF funded? A conference may not be enough, and this is not something that NSF supports. It's difficult for geographically-dispersed groups. Communication needs to cover where the hardest problems are, not just the successes. There is no incentive to talk about negative results. NSF could help here.

Public money is required, and is hard to come by. There is money from the Department of Education for both study and implementation, but you have to think both in terms of long term and short term. There has to be a non-government, non-market entity that can help coordinate this enormous amount of capital with real need. (This is what code.org is attempting to do.) In general, we need to bring together the CS education research community with the people who have money but do not have much CS education research experience.

Often funding institutions favor research, but do not fund the maintenance of the tools for implementation because it is not new work. Could this be reframed as research, as in: what organizational systems and what supports need to be put in place in order for implementation tool creators to maintain their tools?

Data sharing is useful, but there are hurdles involved in sharing student data. Data may need to be destroyed if it cannot be anonymized. National Science Foundation grantees are getting requests to manage data, and repositories are being created. It's useful to remember that data is only as good as the instruments.

We want to get graduate students invested in doing computer education research. It's starting to happen, and there are places where students can obtain a PhD in computer science education research.

What Can CS Learn from Other STEM Disciplines?

How do the economics of plentiful, high paying CS jobs make CS different than other STEM disciplines? Are the methods of teaching computational thinking that are practiced by other STEM teachers qualitatively different than the methods of formally trained computer scientists? Has it been more effective to have a separate engineering education department, or to integrate education with the rest of engineering?

Because CS is a relatively new field, it can learn from older STEM disciplines about what educational practices have been successful. Presentations about physics and engineering education provided a discussion starting point. It was pointed out that compared to other STEM disciplines, CS has an advantage in that it does not have to undo centuries of less effective educational practices.

Carl Weiman of Stanford University gave a keynote about what has been successful in physics education. He attributes its success to using highly regarded physicists backing rigorous research, which involves controlled experiments, assessment data, and replication. They focused the research on teaching a small number of concepts. Since there is more than one type of physics expertise, the solution has been to focus on the commonalities. Dissemination of the research is accomplished through talks at annual meetings by prominent physicists. Physics educational researchers are able to maintain credibility despite specializing in education because they were previously prominent in physics. It also helps that there is a journal devoted to education.

Lance C. Pérez of the University of Nebraska Lincoln gave a history of engineering education in the United States, which spanned 200 years. Key early decisions included one in 1802 to follow the French model of formal education, rather than the English model of apprenticeship, and the Morrill Land Grant Act and the growth it stimulated in the number of institutions with colleges of engineering. Engineering education was also impacted by the Mann report in 1918 and the Grinter report in 1955.

With respect to engineering education research and scholarship, the National Science Foundation's investment in several multi-institution coalitions to perform foundational research on engineering education built the foundation of the current engineering education research community. He described an "arc of utility," where universities do basic and applied research, and the industry does development and use and asked if this was also true of engineering, and more general STEM, education research. He also posed the question as to whether Is it more effective to have a separate engineering education departments, or to integrate engineering education research within the rest of engineering using the disciplinary based education research model?

In some ways, CS is different than other STEM fields. Economics is a big driver, since jobs in CS are plentiful and well-paid. This means that women and minorities are strongly affected economically if there is a lack of CS opportunities for them. Also, computing is a "literacy" that is directly applicable to other STEM fields. (Although, it's worth noting that other disciplines also make this claim.)

In addition to looking at mainstream STEM practices, it is also worthwhile to investigate "folk programming practices", that is ways of learning computational thinking that are practiced by other STEM disciplines, such as chemistry and biology. Are they qualitatively different than formally trained computer scientists?

Bias

Top tier universities are finding that lack of CS teachers is a larger problem than lack of students taking CS, but is this true of most universities? The elite have a passion for the field – how does this affect the expectation that everyone successful in CS must have that passion? What are the values that affect research? What research directions are being missed? Can we create developer personae that do not reflect bias (as compared to the developer personae created at Microsoft)? How do we make entry points into CS more inclusive, accounting for variations, and not creating groups that are too encompassing (such as "people of color")?

Bias came up as a theme several times. At one point, several people began discussing whether there was bias within the conference itself, and a breakout group was formed on the subject of bias. In addition, Josh Tenenberg of the University of Washington Tacoma presented on the subject, asking what questions aren't we asking? He discussed how theories as both lenses and blinders, and how when we choose research projects, our own value systems and priorities go into them.

Several areas of bias were discussed, including:

Inclusiveness. The talk about the crisis at Stanford may only apply to Tier 1 universities, and is very different than the crisis that the vast majority of students are not learning any CS. The conference has focused more on the upper level of education and less on the majority of schools. There is certain language that is used, such as the "caliber" and "potential" of students that is not inclusive. Love and passion for the field is overrated and overstated; that's not necessarily the reason why people do programming. However, the elite do have the passion, and are setting the goals and agenda.

Research. The world as we see it influences our research, and our choice of research affects our values. We need to make values explicit when talking about research. An example of how values influence research is that in physics education, values affected who was chosen to be the experts. Choosing experts was necessary so that the results would be taken seriously, but the bias must be considered. The goal of this summit is to bring leaders together to chart new directions, but the reality is that there is a large range of directions. There was a lot of interest in which papers were not accepted to the conference – what was missed?

Developer Personae. Microsoft has created three personae that represent developers: Mort, Elvis, and Einstein. However, these personae don't represent many types of programmers, for example students coming from small HBCUs or community colleges.

Identity. An interesting activity with students is to have them write their identities (religion, race, gender, language, and two others) on post-it notes. Have them give up one identity at a time. It was surprising what students chose to give a way or hold on to.

Addressing biases. We need to acknowledge and address our biases. This affects who you work with, or how you treat students. For example, you may resent students with disabilities because they are more work for you. Be careful not to lump individuals together; this includes “white men.” We need many entry points into CS, and research and design needs to reflect this, not just lump together “people of color.”

Diversity. A common bias is a belief that diversity does not result in quality, essentially that increasing diversity results in watering down quality. However, if we make entry points inclusive, then we will get more people, and more high-performing computer scientists.

Professional Development

What motivates teachers to pursue professional development? For K-12, how can training occur at both the pre-service and in-service level? What is the impact of budgetary cuts, external requirements (federal, state, and district), and the push for high-stakes testing? What makes CS teachers stay, as opposed to leave for higher-paying industry jobs?

Professional development is a key part teaching CS effectively. The issues are different for higher education compared to K-12. For higher education, for some universities there are not enough undergraduate programs for students interested in computer science. Furthermore, enrollment is drastically increasing because non-major enrollment is increasing. This makes it difficult for faculty to find time for professional development.

For K-12, in most cases, CS teachers don’t get CS training at the K-12 level. This training is needed at both the pre-service and in-service level. There are issues of equity – some schools have more funding for professional development. Budgetary cuts have a large impact on CS teachers. Compared to STEM, CS is considered a vocational subject, and CS teachers get eliminated because CS isn’t considered core education, and may not count towards graduation requirements.

Other challenges include having to deal with policies based on federal, state, and district requirements, the push for high-stakes testing, and requirements and teacher certification are often disconnected. Also, the Hour of Code increased the demand for CS teachers, but teachers with CS skills often leave for industry, because it provides higher salaries.

Nonetheless, there are many reasons that teachers would want to pursue professional development, including job security, passion, forming an identity as a computer science teacher, empowerment to do their job better, aligning with standards and policies, and obtaining resources to use in the classroom.

What is the role for CS education research beyond the first year courses?

What are good pedagogies for junior and senior level undergraduate CS courses? How can pedagogies from other disciplines (for example, inquiry and project-based learning) be used? Who is teaching first year courses compared to later courses? What tools can be developed that can help teachers know what to do and what not to do?

There is a great deal of CS education knowledge beyond the first year. However, fragmentation exists between the different pockets of computing education.

What are good pedagogies for junior and senior level undergrad CS courses? Research should focus on new pedagogues instead of focusing on concepts, but current studies about different pedagogical tools are conflicting. There is sometimes disdain between different individuals based on their teaching practices, and a lack of standardization across schools.

Other disciplines may provide some ideas. For example, science classes often do inquiry and project-based learning.

It is also worthwhile to think about who is teaching first year courses compared to later courses. People with CS education degrees often end up teaching first year courses, and graduate students are often required to teach upper division courses. It would be valuable for undergraduates to teach.

There are many tools that can help teachers know what to do and what not to do. Some ideas include: a checklist manifesto (like in surgery), recording problem-solving with live scribe, having teachers identify what students should be able to do at the end of the lecture, creating a short video of bad teaching practices, and creating an extreme makeover reality TV show.

Next Steps / Conclusion

The meeting proved to be a good community building exercise, and the enthusiasm of the participants supported the desire and need for more discussion. While there was not universal accord on all topics, there was a unified desire to continue developing/growing a community and indications were that next steps would be eagerly received. Hopefully, the CER community will agree to come together for future research discussions, perhaps at five-year intervals.

This January 2014 workshop focused on determining important research questions CER should be addressing. There are at least three take-aways.

First, we need to deal with structural issues in supporting CER in universities. How do we encourage research universities to support researchers, faculty, and especially graduate students doing CER? This topic will be the focus of an upcoming meeting to be held in March 2014.

Second, it is still unclear how the community will prioritize research questions and whether we should be identifying grand challenges. While we did largely come together to agree on the general research areas that are most pressing for CER, we did not identify specific research questions.

A third topic is that there are at least two conferences focusing on CER research - ICER and Learning@Scale. These conferences are largely independent of one another and, at present, are not encouraging the community to develop one clear unified voice. While the group at this summit consisted of faculty involved in both conferences, it was certainly dominated by those more closely involved with ICER. More work needs to be done towards involving both communities in future CER discussions.

References

Sylvia Scribner, Literacy in Three Metaphors, The American Journal of Education 93 (1984) 21. 1984 by University of Chicago Press.

Basawapatna, A. Repenning, A. Poster Cyberspace Meets Brick and Mortar: Finding the Sweet Spot between Cyberlearning and Traditional Instruction. Poster Presented at SIGCSE '10. (Milwaukee, WI, March 10-13, 2010).

Cuny, J. 2010. Finding 10,000 teachers. CSTA Voice, 5, 6, (Jan. 2010), 1-2.

Engle, R. A., Roberts, S., Nguyen, P. D., Yee, P. & the Framing Transfer Research Group (2008). A design-based approach to experimental design: Investigating hypotheses about how framing influences transfer. Proceedings of the International Conference of the Learning Sciences.

S. Turkle and S. Papert, Epistemological Pluralism and the Revaluation of the Concrete, SIGNS: Journal of Women in Culture and Society, Autumn 1990, Vol. 16 (1)

Research Design: Qualitative, Quantitative, and Mixed Methods Approaches, John Creswell, SAGE Publications, Inc; Fourth Edition edition (March 14, 2013)

Wing, J. 2006. Computational thinking. Commun. ACM 49, 3 (March 2006), 33-35.

Appendix A – Summit Agenda

WEDNESDAY	January 8, 2014	
1:00-1:30	Registration and Snacks	
1:30-1:45	Welcome and Overview of the Summit	Steve Cooper
1:45-2:30	Keynote Address: The Future of Computer Science Education Research - Perspectives and Lessons from Physics Education Research and Cognitive Psychology	Carl Wieman
2:30-3:00	STEM Education	
2:30	Computational Thinking through Modeling and Simulation	Uri Wilensky
2:40	CS Education and Engineering Education: Joint Research to Develop Students' Interest and Success	Johannes Strobel
2:50	Discussion	
3:00-3:30	K-12	
3:00	Professional Development for CS Teachers: A Framework and its Implementation	Aman Yadav
3:10	How can we help a more diverse range of young people become fluent with computational technologies?	Mitchel Resnick
3:20	Discussion	
3:30-4:00	Broadening Participation in Computing	
3:30	Increasing the Participation of Women and Underrepresented Minorities in Computing: Missing Persepectives and New Directions	Catherine Ashcraft
3:40	Nonvisual Access to Computation and Programming	Clayton Lewis
3:50	Discussion	

4:00-4:30	Break	
4:30-5:00	Students and Learning	
4:30	Building Evidence and Building Practice in Computer Science Education	Marie Bienkowski
4:40	Reframing Research Questions in CS Education: Unpacking the Suppositions About Mind and Activity	Josh Tenenberg
4:50	Discussion	
5:00-5:30	Structure and Policy	
5:00	Developing Research-Informed Instructional Resources to Support Broad Scale Access to High Quality Computer Science Education: A Research Agenda	Jeanne Century
5:10	Computing Education Research	Wil Doane
5:20	Discussion	
5:30-6:30	Breakout Sessions 1) How could we demonstrate impact (if any) of learning computer science on learning and performance in other STEM fields? 2) As we move from computer sciences as a vocational subject to computing as a literacy that is desirable for all professionals, how do we need to restructure our classes and our education organizational structures to support this shift? 3) What influences teacher decisions to pursue (or more likely, not pursue) professional development towards teaching computing? 4) If we ground students in concrete and contextualized ways of learning, how do they learn more abstract ways of thinking and doing? How do they lift above context and concreteness? 5) Can anyone learn to program? Is this an innate bar that some cannot clear?	
6:30-7:30	Dinner and Keynote Address: Teaching All Students Computing	Eric Roberts

THURSDAY	January 9, 2014	
8:30-9:00	Breakfast	
9:00-9:45	Keynote Address: A History of Engineering Education and Lessons Learned	Lance Pérez
9:45-10:15	Report on Breakouts from Wednesday Sessions	
10:15-10:30	Break	
10:30-12:00	Breakouts 1) What are student beliefs about the computing disciplines, and how does this affect their choices about further pursuing one or more of these disciplines? 2) Are there developmental constraints for learning different aspects of programming at different points in human development (as has been hypothesized for mathematics and other formal kinds of reason)? 3) Will the further penetration of computing in K-12 provide legitimization and wider practice of Computing Education Research, in the way that it has done for Math and Science Education Research? 4) To what extent is computer science similar to and/or different from the other STEM disciplines when considering issues of equity, and more broadly of under-representation?	
12:00-1:00	Working Lunch with Topic Generation Activity	Mark Guzdial
1:00-1:30	Report on Breakouts from Thursday morning session	
1:30-2:15	Keynote: Learning	Sally Fincher
2:15-3:15	Breakouts: Topics TBD by Topics Generated Earlier	
3:15-3:30	Snack Break	
3:30-4:00	Report on Breakouts from Thursday afternoon session	
4:00-4:45	Discussion of Future Directions Summit Report	
4:45-5:00	Summit Wrap-Up	Steve Cooper

Appendix B: Summit attendees

Name	Institution
Catherine Ashcraft	University of Colorado Boulder
Yifat Ben-David Kolikant	Hebrew University of Jerusalem
Marie Bienkowski	SRI International
**Linda Bookey	Bookey Consulting
Kristy Boyer	North Carolina State University
Karen Brennan	Harvard University
Quincy Brown	Bowie State University
**Engin Bumbacher	Stanford University
Michael Caspersen	Aarhus University
Jeanne Century	University of Chicago
*Steve Cooper	Stanford University
Brian Danielak	University of Wisconsin – Madison
Wanda Dann	Carnegie Mellon University
Jill Denner	ETR Associates
Betsy DiSalvo	Georgia Institute of Technology
William Doane	Science & Technology Policy Institute
Stephen Edwards	Virginia Polytechnic Institute and State University
Michael Eisenberg	University of Colorado Boulder
Sally Fincher	University of Kent
Kathi Fisler	Worcester Polytechnic Institute

***Jeff Forbes	Duke University / National Science Foundation
Christina Gardner-McCune	Clemson University
Joanna Goode	University of Oregon
**Shuchi Grover	Stanford University
**Peter Gruenbaum	Bookey Consulting
*Mark Guzdial	Georgia Institute of Technology
Apriel Hodari	Council for Opportunity in Education
Christopher Hundhausen	Washington State University
Yasmin Kafai	University of Pennsylvania
Barry Kurtz	Appalachian State University
Richard Ladner	University of Washington
***Trey Lathe	NSF
Cynthia Lee	Stanford University
Clayton Lewis	University of Colorado Boulder
Colleen Lewis	Harvey Mudd College
Jane Margolis	University of California, Los Angeles
Robert McCartney	University of Connecticut
**Kathy Menchaca	Stanford University
Kai Orton	Northwestern University
Lance Pérez	University of Nebraska - Lincoln
**Chris Piech	Stanford University
Enrico Pontelli	New Mexico State University
***Jane Prey	National Science Foundation
Alex Repenning	University of Colorado Boulder

Mitchel Resnick	MIT
Eric Roberts	Stanford University
Susan Rodger	Duke University
Mehran Sahami	Stanford University
*Beth Simon	University of California, San Diego
Leen-Kiat Soh	University of Nebraska
Sheryl Sorby	The Ohio State University
Andreas Stefik	University of Nevada Las Vegas
Johannes Strobel	Texas A&M University
***Harriet Taylor	NSF
Josh Tenenberg	University of Washington, Tacoma
***Paul Tymann	NSF
Carl Wieman	Stanford University
Uri Wilensky	Northwestern University
**Marcelo Worsley	Stanford University
****Aman Yadav	Purdue University

*Summit PI
**Summit Staff
***NSF Program Officer
****Unable to attend due to weather conditions