Lightning Talk Abstracts

ICER 2013

We are once again pleased to have a collection of Lightning Talks as part of the ICER conference program. These talks are short, strictly timed presentations intended to further expand the ICER community and spark discussion among conference participants. The goal is for these talks to provide a venue in support of both new ideas and newcomers to our community.

Like the main ICER program, the Lightning Talks cover a wide range of computing education related work, and the presenters are from a variety of different institutions around the world. Short abstracts for each of the lightning talks to be presented at this year's conference can be found on the pages to follow.

Brian Dorn
2013 Lightning Talks Chair
University of Nebraska at Omaha, USA

Lightning Talk Presentation Order

1. *Helping Kids Learn Game Design through Instant Feedback*
   James Counts
   California State University San Marcos, USA

2. *Observing thousands of programming learners*
   Neil Brown
   University of Kent, UK

3. *Activate Computational Thinking (ACT)*
   Katherine Hayden
   California State University San Marcos, USA

4. *OSBIDE: A Social Programming Environment to Support Awareness and Collaboration within Individual Programming Assignments*
   Adam S. Carter and Christopher D. Hundhausen
   Washington State University, USA

5. *"You're too much of a people person to be a computer scientist": How students feel they fit (or don't fit) in CS*
   Colleen M. Lewis
   Harvey Mudd College, USA

6. *Improving Learning and Confidence in Senior Database Systems*
   Edwin M. Knorr
   University of British Columbia, Canada

7. *How to Grade an Outcome-Based Grading System?*
   Kathi Fisler
   Worcester Polytechnic Institute, USA

8. *Code Construction Pathways: Understanding Student Attempts*
   Leigh Ann Sudol-DeLyser
   New York University/Carnegie Mellon University, USA

9. *Student Participation in Khan Academy Computer Science*
   Youwen Ouyang
   California State University San Marcos, USA

10. *YouSpeak: A Classroom Interaction Software System*
    Ani Nahapetian
    California State University Northridge, USA

Helping Kids Learn Game Design through Instant Feedback

James Counts
Department of Computer Science and Information Systems
California State University San Marcos
count009@cougars.csusm.edu

Abstract:
Game systems provide an opportunity to discover an interest in computer science and programming [1]. Modern 3D games, however, are built on sophisticated programming platforms. An interested child would need to work with trigonometry, pointer manipulation and real-time programming in order to manipulate newer systems. By the time children learn these concepts, which would likely be in college, they may very well have rejected computer science (or STEM) as a career path.

Our system aims to lower the barrier to entry for game design, because designing a good game is about more than learning how to program a specific language or platform. Game design involves thinking about rules that work together, and manipulating ideas in order to create a consistent experience. Visionaries like Bret Victor [2] have demonstrated techniques effective in reducing the need to rely strictly on written programming languages in order to manipulate digital media. Our research creates an interactive environment to teach children to think like a game designer instead of a programmer. The interactive environment provides instant feedback for students as they adjust elements of a platform game such as board/world dimension, agility of the player, distribution of platforms, etc. This talk will present analyses of students' interaction with the system.

[1] DiSalvo, B. J. and Bruckman, A. 2009 "Questioning Video Games' Influence on CS Interest", in Proceedings of the 4th International Conference on Foundations of Digital Games, pp 272 – 278 April 26 – 30, 2009, Orlando, FL

[2] Victor, Bret. "Magic Ink: Information Software and the Graphical Interface." Released on March 15, 2006, available at http://*worrydream.com*.

Observing thousands of programming learners

Neil Brown
University of Kent
nccb@kent.ac.uk

Abstract:

Last year, at ICER 2012, we described a plan to augment the beginners' IDE, BlueJ, to record data about the activities of BlueJ users [1]. That plan has now come to fruition, and the latest release of BlueJ (3.1.0) includes an opt-in prompt to join the data collection research. The potential scale of the data collection is already apparent; within the initial three days after release, we gained more users than the 1,101 in the Edwards et al study [2], the largest such prior study that we are aware of.

This lightning talk will give a brief summary of the early data collection outcomes, such as the user numbers, opt-in rate and the amount of data being collected. There will also be a reminder of how other researchers can get access to the data to use it in their own computing education research, thus benefiting the whole computing education research community.

[1] I. A. Utting, N. C. C. Brown, M. Kölling, D. McCall, P. L. C. Stevens. 2012. Web-scale Data Gathering with BlueJ. In *Proceedings of the eighth international workshop on Computing education research (ICER '12)*. ACM, New York, NY, USA.

[2] Stephen H. Edwards, Jason Snyder, Manuel A. Pérez-Quiñones, Anthony Allevato, Dongkwan Kim, and Betsy Tretola. 2009. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the fifth international workshop on Computing education research workshop (ICER '09)*. ACM, New York, NY, USA.

Activate Computational Thinking (ACT)

Dr. Katherine Hayden
CSUSM
khayden@csusm.edu

Abstract:
There is a lack of research on what it takes for content subject teachers to gain Computational Thinking (CT) proficiencies. In addition, there are no documented professional development models to support subject area teachers in designing student CT learning experiences within their disciplines.

In 2010, the Leveraging Thought Leadership for Computational Thinking in K–12 Curriculum project funded by the NSF published CT Teacher Resources that include the operational definition of CT, a continuum chart, and nine CT learning experiences, as well as CT classroom scenarios.

In 2011, new K-12 Computer Science Standards were developed by the CSTA Standards Task Force providing a 3-level framework for computer science, with level 2 targeting grades 6 – 9 to ensure middle school students "begin using computational thinking as a problem-solving tool" through learning experiences "embedded in other curricular areas such as social science, language arts, mathematics, and science" (p. 13).

ACT is an NSF funded *research* project focusing on effective professional development for teachers to increase understanding of CT and the integration of CT activities into core science curriculum in middle school classrooms. Through intensive summer academies followed by ongoing professional development and mentoring, as well as participation in a professional learning community throughout the school year, the ACT project is closely observing and studying how teachers develop CT competencies and apply appropriate pedagogies as they design and implement standards-aligned CT activities in their science classrooms. This talk will share findings from the 2013 ACT summer academy.

OSBIDE: A *Social* Programming Environment to Support Awareness and Collaboration within Individual Programming Assignments

Adam S. Carter and Christopher D. Hundhausen
Human-centered Environments for Learning and Programming (HELP) Lab
School of Electrical Engineering and Computer Science
Washington State University
{cartera, hundhaus}@wsu.edu

Abstract:
Especially in early computing courses, students are commonly expected to work on programming assignments individually, outside of a shared lab space. This practice can lead to social isolation and lower programming self-efficacy—factors known to reduce persistence within the discipline. Moreover, this practice runs counter to common programming practices in industry, where software engineers routinely collaborate on team programming projects.

In this lightning talk, we introduce a possible way forward that leverages current trends in social networking. OSBIDE (Online Studio-Based Integrated Development Environment) represents a new breed of *social programming environment* that enables students in a class to participate in an online community rooted in shared programming activities and progress. OSBIDE promotes social awareness and collaboration by compiling students' programming activities (compilation attempts, compilation errors, run-time errors, debugging events, and the like) into a shared *activity stream*. This stream, viewable by all, can be searched and commented on in a manner similar to that of popular social networking sites such as Facebook.

Whereas other collaborative software development environments are designed to support teams as they work on a shared code solution, OSBIDE supports *individual* assignments in which each student implements his or her own solution to the same programming problem. This approach favors the traditional approach taken in early computing courses, which typically emphasize individual work.

OSBIDE is presently being piloted in a CS 1 course. As we continue to develop OSBIDE iteratively, we will need to address several key questions, including (1) What programming activities will be most useful to users?, (2) How can programming activities be best presented in the activity stream so as to make them relevant and meaningful to users?, and (3) How will students participate in the environment in order to build community and learn about computer programming?

"You're too much of a people person to be a computer scientist": How students feel they fit (or don't fit) in CS

Colleen M. Lewis
Harvey Mudd College
lewis@cs.hmc.edu

Abstract:

Despite recent increases in undergraduate enrollment in computer science (CS), degree production might still fall short of projected industry demand. This lightning talk will provide a glimpse of our continuing research on undergraduates' decisions to major or not major in CS. This work is based on interviews conducted with a total of 31 introductory programming students at two U.S. institutions. A previous paper reported on the role of students' CS self-efficacy in their choices of major [1]. Current analysis focuses on another factor that appears to affect decisions to major in CS: a student's perception of whether they belong or can "fit" in CS.

Recognizing that the notion of fit is subjective, our analysis seeks to understand how students arrive at conclusions concerning their fit in CS. The students' accounts show that determinations of fit incorporate information from cultural stereotypes, their observations, and their direct experiences related to CS, both in and out of the classroom. More specifically, students considered whether it was necessary to be competitive, asocial, male, and/or singularly focused on programming/CS in order to fit in CS. This is not to say that most students believe that you need to have all of these characteristics to major in CS. On the contrary, interviewees described different ways to wrestle with and reconcile their identity, interests, and expectations of what is required to fit in CS. Frequently, this negotiation took place in a complex context, including peers, instructors, advisors, family, media, and more. We hope that better understanding how students think about fit will inform CS educators' efforts to welcome, support, and graduate a broader, more engaged set of students.

[1] Colleen M. Lewis, Ken Yasuhara, and Ruth E. Anderson. 2011. Deciding to Major in Computer Science: A Grounded Theory of Students' Self-Assessment of Ability. Proc. ICER 2011.

Improving Learning and Confidence in Senior Database Systems

Edwin M. Knorr
University of British Columbia
knorr@cs.ubc.ca

Abstract:

Our course in advanced database systems deals with the internals of relational database systems, including performance issues, the I/O cost model, indexing, metadata, external sorting, query evaluation, and query optimization. Essentially, we are opening the black box of database systems to see how the pieces fit together. We also include business intelligence topics such as data warehousing (e.g., schema design, data cubes, view materialization) and data mining (e.g., KDD process, frequent itemsets, association rules, recommender systems). The number of topics and learning goals is large.

Students tend to spend a lot of time on the assignments (homework), and are often frustrated by the number of details or interacting parts—many of which are not apparent until they see the detailed solutions. Assignment marks for this non-programming course tend to be consistently lower than those in other CS courses.

We conjecture that students would benefit from seeing lots of worked examples at various placements/categories within Bloom's Taxonomy [1]. We plan to: (a) provide students with *many* worked examples [2]; (b) increase the number of midterms from two to four per semester, while reducing the number of assignments to zero; and (c) hold students accountable for practicing the worked problems, by giving grades for clicker questions and other in-class exercises. Learning and confidence will be measured by comparing outcomes (and surveys) to previous course offerings.

[1] Anderson, L. & Krathwohl, D. A. (eds.) 2001. *Taxonomy for Learning, Teaching and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*, New York: Longman.

[2] Kirschner, P.A., Sweller, J., & Clark, R.E. 2006. Why minimal guidance during instruction does not work: An analysis of the failure of constructivist, discovery, problem-based, experiential, and inquiry-based teaching. *Educational Psychologist*, 41(2), 75-86.

How to Grade an Outcome-Based Grading System?

Kathi Fisler
WPI Department of Computer Science
kfisler@cs.wpi.edu

Abstract:

Two years ago, frustrated by a lack of information about what students in my large CS2 class were actually learning, I introduced a new grading system. I identified 40-odd learning outcomes that I thought the course covered: each was one of factual (e.g., "know the definition of a hash table"), foundational (e.g., "know how to create a class"), or conceptual (e.g., "know how to test a function that can produce multiple correct answers"). The grading rubric for each assignment, quiz, and exam earned students points towards individual outcomes. Final course grades were based on mastery of the outcomes, with more weight on the conceptual outcomes. Students had multiple chances to earn points towards nearly all outcomes, and could stop working on problems related to an outcome once they had mastered it.

In theory, this system should have several advantages over a traditional system that gives a single numeric score to each assessment: students should get more nuanced feedback about their progress, I should have a better sense of what students are (not) learning from each assessment, and I should be able to draw more meaningful distinctions between students in terms of final grades. In practice, this still-evolving system has been challenging and time-consuming to implement and administer. Is it worth it?

I am exploring how to assess this grading system, both standalone and relative to my former system. Getting subjective student feedback is straightforward, but what are the right questions to ask? What objective questions should one ask when grading a grading system? What data would help reveal the cost-benefit tradeoffs? How can we move beyond "did we like it" to answer useful questions for the Computing Education Research community?

Code Construction Pathways: Understanding Student Attempts

Leigh Ann Sudol-DeLyser
New York University/Carnegie Mellon University
lsudol@cs.cmu.edu

Abstract:

While computer science is not the practice of programming, the act of writing code is a large part of introductory courses in computer science. There are many tools and environments that are designed to support the novice in learning the skill of writing code. Many of these environments capture the attempts made by students as they progress from the start to the point where they exit the question, either with a correct answer or a incomplete attempt. There is rich data in the submissions made by students, however the codification and analysis of such data is time consuming and difficult. Especially in environments where students are constructing algorithms with multiple components, there are few tools and methods for using the code to infer student knowledge about the components of the problem.

In this lightning talk I will discuss the automatic codification of student code submissions through a static analysis technique and the resulting data analysis to understand student learning across multiple problems. Students interacted with a system where they wrote code to implement four methods in the Java programming language. The methods each required the student to iterate over an array to complete an algorithm such as a max search. The student submissions were codified to indicate the correct use of an algorithmic component, such as a for loop. Using all of the submissions for a single problem, a graph was created which modelled student progression through the code writing and debugging process. The resulting model provides a rich view of student problem solving techniques and common paths to a correct solution.

Student Participation in Khan Academy Computer Science

Youwen Ouyang
Department of Computer Science and Information Systems
California State University San Marcos
ouyang@csusm.edu

Abstract:

Khan Academy Computer Science (KACS, https://www.khanacademy.org/cs), according to its architect John Resig, is "a platform that targets people with no programming knowledge and give them an engaging and fun environment to learn in" [1]. The platform provides a two-panel view with the left being an Ace editor (http://ace.ajax.org) for students to experiment with their code and the right a canvas to show the result of students' code. Inspired by Bret Victor's talk on responsive programming environment [3], KACS allows students to immediately see the result of their code as they change it. A number of tutorial videos are provided by KACS to cover fundamental programming concepts. For each tutorial, students can pause the video anywhere, which switch the video panel to the editor and students can start adjusting the code to see its result. Students can resume the video when they are done with their code experiment. In addition KACS supports a collaborative community of learners by mimicking open source programming communities like Github. Students can "fork" any project into a spinoff of the project that can be modified and further extended.

This talk will present some analyses of student participation data that have been collected using Python to parse the tutorials and spinoff pages on KACS. The analyses look for patterns showing how different tutorials are used by students of different programming background and how students contribute to the KACS community. The presenter hopes to engage in discussions with computing education researchers about potential follow-up studies.

[1] Resig, John 2012. *Redefining the Introduction to Computer Science*. A Blog posted on August 14, 2012 and available at http://ejohn.org/blog/introducing-khan-cs/.

[2] Victor, Bret 2012 *Inventing on Principle* A talk given at The Canadia University Software Engineering Conference (CUSEC 2012) and available at http://vimeo.com/36579366

YouSpeak: A Classroom Interaction Software System

Ani Nahapetian
California State University, Northridge (CSUN)
ani@csun.edu

Abstract:

YouSpeak [1] is a web-based and mobile classroom interaction software system, developed and used at California State University, Northridge. It provides a framework for students to comment and ask questions anonymously during class, using the web or a web-based app. Instructors project these comments alongside the course materials (such as slides or the marker board) using the web interface. Moreover, students can rate up or down the comments of others, to allow the most relevant questions to rise to the top of the list.

The project's goal is to increase access to lecture participation opportunities for students who may find traditional approaches challenging (even only at certain instances in time). The continuous and personalized data collected about student understanding of course topics tailors the immediate course flow, and also informs the courses in the long term. The corpus of data produced can be used to extract contributing factors to student participation and the understanding of course topics.

The well-known technology of clickers allowed students to give electronic feedback during lecture. YouSpeak goes a step further by not only examining the understanding of concepts by the class as a whole, but also the understanding of concepts by certain groups of students in the class. For example, a topic presentation that may be understood differently by the hearing-impaired can be determined, and a more inclusive approach can be considered. Additionally, the system gives students a more flexible communication channel during the lecture.

We hope to have the lightening talk inspire feedback regarding the tool's features, and also to encourage collaborators to experiment with the tool on their own campuses. Additionally, we plan to summarize the results of a trial run in a recent Software Engineering course offering.

[1] YouSpeak website. http://www.ecs.csun.edu/youspeak.