Using a Discourse-Intensive Pedagogy and Android's App Inventor for Introducing Computational Concepts to Middle School Students

Shuchi Grover
Stanford University
Graduate School of Education
Stanford, CA 94305
shuchig@stanford.edu

Roy Pea
Stanford University
Graduate School of Education/
H-STAR Institute
Stanford, CA 94305
roypea@stanford.edu

ABSTRACT
Past research on children and programming from the 1980s called for deepening the study of the pedagogy of programming in order to help children build better cognitive models of foundational concepts of CS. More recently, computing education researchers are beginning to recognize the need to apply the learning sciences to develop age- and grade-appropriate curricula and pedagogies for developing computational competencies among children. This paper presents the curriculum of an exploratory workshop that employed a discourse-intensive pedagogy to introduce middle school children to programming and foundational concepts of computer science through programming mobile apps in App Inventor for Android (AIA).

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer and Information Science Education–*computer science education, curriculum, literacy*

General Terms
Design, Human Factors, Languages.

Keywords
App Inventor, Android, Computational Thinking, CS Ed Research, Experience Report, Introductory Programming, K-12 Instruction, Middle School.

1. INTRODUCTION
Wing's articles [22, 23] have been influential in arguing for adding computational thinking (CT) to every child's analytical ability as a vital ingredient of STEM learning in K-12. The tailwinds in the larger environment have further fanned this belief. The imperative for computing education in K-12 has gained momentum not only following alarming reports [21] but also due to the surging interest in STEM learning since the turn of the 21st century. There now appears to be growing consensus around the view that all children must learn CT and be offered a robust introductory exposure to computer science in K-12. Globally as well, countries around the world are beginning to act on the rationale for introducing computing education as early as middle school in order to train young minds in this discipline and way of thinking.

How do children best learn computational concepts? Seymour Papert's pioneering efforts in the 1980s around children, programming, and the development of procedural thinking skills through LOGO programming [16, 17] inspired a large body of research studies. This previous literature on children and programming (such as [7], among others) revealed the types of problems children experience on their way to understanding computing, and overwhelmingly called for a need to study the pedagogy of programming in order to help children build better cognitive models of foundational concepts of CS. However, most recent research in computing education and CT in K-12 that has used Wing's article as a springboard, has focused less on *pedagogy* and *process*, and more on *tools* for CT development, and learner-created programming *artifacts* to assess development of CT. Thus, despite the flurry of recent research activity on CT, many key questions still remain unanswered, and there is much that needs to be done to help develop a more lucid theoretical and practical understanding of how children come to understand computational concepts and thus how best to design the teaching and learning experience [5].

Pertinent to this context, it is a subsequent call from Jeannette Wing that deserves more attention than it has received thus far. At the "Workshop of Pedagogical Aspects of Computational Thinking," convened by the National Academy of Science, Wing argued for an application of research in the sciences of learning to design grade- and age-appropriate curricula for CT to maximize its impact on and significance for K-12 students [14]. In a view echoed by Alfred Aho at the same workshop, Wing acknowledged that the application of the learning sciences to fields like math had helped successfully develop learning progressions that have a solid foundation in research on the human brain and how it enables the learning of mathematical concepts. In contrast, computing has thus far been introduced to children in K-12 without much thought of how children will best learn CS concepts.

Key research findings in the learning sciences in the past couple of decades have been centered on learning as a *social endeavor*. These have included investigations of cognition and learning as it occurs in *socio-cultural contexts*, with all the attendant ideas of situated learning, distributed intelligence, cognitive apprenticeship, embodied cognition, as well as activity, interaction and discourse analysis. Computing education curricula and pedagogy, as well as recent research in CT, have thus far not really leveraged these new developments on how children learn.

2. CLASSROOM DISCOURSE

The work presented in this paper draws on the well-known Vygotskian theoretical framework that emphasizes the importance of social interaction in the development of individual mental processes. Learning is a social activity, and speech unites the cognitive and the social [1]. Thus, in order to foster the development of any competency in a social context inside or outside the classroom, it is crucial that the pedagogy and curriculum pay attention to designing communication activities so that they foster better student learning. Learning scientists believe that in the process of guided knowledge construction*, tasks, representational tools, and talk are inextricably intertwined* [20] and that "sense-making and scaffolded discussion, calling for and elicited by particular forms of talk, are seen as primary mechanisms for promoting deep understanding of complex concepts and robust reasoning." Recent research on classroom discourse especially in math and science classrooms has revealed learning gains and better understanding through the use of discourse-intensive pedagogical practices where tasks are combined with interactions among learners, and productive teacher-led discussions.

3. PILOT AIA WORKSHOP FOR MIDDLE-SCHOOL STUDENTS

3.1 Designing for Discourse in a Computational Setting

Different domains and settings call for different types of tasks, discourse and activity [9]. The workshop pedagogy described here took the novel approach of bringing ideas of classroom discourse from the learning sciences to the teaching of introductory computational concepts. The curriculum of this pilot workshop was thus designed around what we would like to call out as ***"Computational Discourse," where learners would be introduced to ideas of computer science through building competencies in computational thinking by knowledge building discussions in concert with engaging in computationally rich activities.*** Based on past research on classroom discourse, the broader goal and belief was that through the process of "academically productive talk" [10], interactions, discussions and collaborations amongst learners and teacher, children would be supported in their "doing programming", making sense of computation concepts, thinking computationally, and developing a new vocabulary in practice. In doing so, learners would at the same time take on a new identity – that of computational thinker and computer scientist – which would build upon and transform their current understanding of computational concepts and ways of speaking about them and, ultimately, how these concepts and competencies relate to the goals that they have and what they imagine themselves doing in the future. Analogous arguments on the synergistic development in knowledge, skills, discourse and identity that learning environments should seek to support underlie the new K12 Science Education Framework and associated K12 Science Education Standards [15].

The following sections describe the pilot workshop, its curriculum, and the learning experience that evolved through teacher-learner discussions. The goals of the workshop (and the larger research on CT that it was a part of) were to understand and describe-

a) How computational discourse played a role in a computationally-rich learning setting that aimed to foster the development of CT, and

b) The appropriateness of App Inventor for Android (AIA) as a tool for fostering discourse and computational learning in such a setting.

3.2 Participants and Activities

Seven middle school students (3 girls, 4 boys; mean age: 13 years) with little to no prior programming experience were recruited to participate in a pilot workshop for middle school children titled 'Build Your Own Mobile Apps'. Since the workshop was part of a broader research to study the process of development of computational thinking in tweens and teens, the students were also informed that this entailed participation in a research study, and IRB-approved assent and consent was sought from the participants and their parents.

The study was conducted over a daylong workshop from 8:30 am to 5:30 pm in a university research lab. One of the authors was the lead researcher for this study as well as the sole facilitator of the workshop. A couple of volunteers provided minor assistance with video capture, and in helping one of the participant groups think through their final app project. Each participant was provided with a laptop and an Android mobile phone device that had been organized for the purpose of the workshop.

For the purposes of research, the following data was collected or captured:

1. The participants filled out brief pre- and post- workshop surveys. The pre survey aimed to gather basic information about the participant's age, grade and prior experience with programming and knowledge of mobile phone apps. The post survey aimed to get a sense for the participants' workshop experience, what they learned during the day, and how their views of what apps are and how they work had evolved.
2. Video capture of the whole room via 2 cameras.
3. Camtasia capture of each of the student laptop screens with picture-in-picture webcam capture of the participants as they worked on their apps.
4. Additional audio capture via recorders placed on the table around which the participants were seated.

3.2.1 AIA Workshop Curriculum

The workshop was divided into two distinct sessions separated by a lunch break. The morning session was spent introducing the participants to the basics of building mobile phone apps using AIA. Since the purpose of the research was to examine a discourse intensive pedagogy for introducing children to programming and foundational computational concepts, the curriculum was designed to be driven by discussion and questions emerging from three introductory collaborative programming exercises that were borrowed from the book, *App inventor: create your own android apps* [24], and performed as a whole group (See Table 1). The facilitator's laptop was connected to a large screen to facilitate discussion and activity involving the entire group, with each participant following along on his/her own laptop and mobile device as well.

Table 1: List of introductory examples

Activity	App and Features
App Example#1 –Hello Purr	Original "Hello Purr" App enhanced to include response to vibration sensor & clock timer events; picture taking and text-to-speech to greet when tapped; modified to have cat or tiger image appear randomly with 'meow' or 'roar' sound.
App Example #2 - Paint	Paint App enhanced to dynamically change radius of dots and width of line; subsequently modified to include features of Example#1 i.e. set canvas background to picture clicked and doodle on it
App Example #3 - Animation	Basic Animation- 2 sprites moving at random around the screen; timer events; coordinate system of phone screen; random number generation

At lunch, students worked in pairs to brainstorm ideas for the apps that they would build based on what they had learned in the course of the morning session. The afternoon was spent programming apps in pairs (except for 1 participant who preferred to work alone).

3.2.2 Final Projects

By the end of the day-long workshop, the participants had developed the following 4 apps that they demonstrated to the group—

- An app to help a child avatar gather all the candy on the screen by dragging him around to each candy. The app kept score of the candy gathered. The participants attempted to incorporate a countdown timer to gather all the candy within a specified time, but ran out of time before they could fully implement that feature.
- A whack-a-mole app that keeps score of the number of moles whacked; (note that the teacher had made passing reference to this game in the morning session, but the students developed it on their own. It was thus quite distinct from the *Mole Mash* example on the AIA website; it used images of moles that popped out of the ground (the background was carefully selected accordingly), and some moles appeared and then disappeared if they had not been whacked (this would impact the score). The students were in the process of developing a second level (after a certain score had been attained), which entailed having the moles appear at a faster rate.
- A bowling app where the goal is to have the ball hit a set of pins moving back and forth horizontally.
- A music jukebox app.

4. EVALUATION

4.1 How did Computational Discourse shape the learning experience?

Qualitative data analysis that involved coding for "discourse moves" in audio and video transcripts indicated the substantial and significant role that discourse–-specifically learner questions and participant actions–-played in the flow of the workshop curriculum. Group discussions significantly influenced the organic introduction and use of new vocabulary as well as important foundational computing concepts that the learners had been previously unaware of.

As an analytic strategy, we also looked for conditions in the context that precipitated learners' inquiry into, and use of, CT. In some instances it was the design invitation of an undesirable outcome or a critical omission that led to productive discourse and introduction of key CT ideas. In all these instances, the facilitator appropriately mediated these opportune "openings" to foster a discussion of key ideas of programming and computing.

Five illustrations of this phenomenon are provided in the subsections below. They describe the scenario that preceded a learner comment or action or question, and the ensuing discourse moves made by the facilitator. While these examples are merely illustrative, they provide clear evidence of how learner and facilitator responsiveness to *emergent issues* at the intersection of task, tools and talk in the course of a discourse-based introduction to simple programming concepts served to trigger opportunities for incorporating more advanced computing concepts, vocabulary, as well as features for learning and solving problems computationally in an organic and meaningful way.

Evidence from Camtasia screen and webcam captures of conversations during the app programming process, as well as pre-post surveys are still being analyzed, but preliminary results reveal a significant growth in CS vocabulary as well as use of important CT elements in the context of building mobile apps.

4.1.1 Illustrative Example #1

Scenario: In the original 'Hello Purr' app, the user taps the image of a cat and hears a 'meow' sound. The facilitator talked about the clock as another way of generating a 'timed' event at regular intervals and asked students to change the app to respond to a clock timer event (every one second) to play the "Meow" sound (instead of the initial design of responding to the user tapping the cat image). This led to the annoying (undesirable) result of 'meow' sounds every second.

Participant Question: "What if I, umm, just wanted the cat to meow five times?

Teacher-initiated Discourse Moves (that influence the curriculum trajectory): Design invitation of an undesirable outcome led to introduction of –

- Concept of keeping count using a 'counter'
- Variables (and explanation of what a 'variable' is)
- Conditional statements and if-then checks to make the program do one thing or another

Analysis: All these were new ideas and associated CT vocabulary terms that were organically introduced, arising from a learner need. These went far beyond the concepts introduced via the 'Hello Purr' app as described in the AIA tutorial.

4.1.2 Illustrative Example #2

Scenario: While the group was modifying the original 'Hello Purr' app as described above-

Participant Action: One participant discovered a way to program the app to take a picture using the phone camera.

Teacher-initiated Discourse Moves (that influence the curriculum trajectory): The facilitator's attention was drawn to this when she heard the click of the phone camera shutter. She decided that the next enhancement to the 'Hello Purr' app would be to have the students take a picture of themselves to replace the original image of the cat and say their name (using the "TextToSpeech" command) instead of playing the 'meow' sound.

Analysis: Two new features were introduced to a whole group as a result of a learner action – using an image clicked using the phone camera as well as the use of the TextToSpeech command.

4.1.3 Illustrative Example #3

Scenario: In the 'Paint' app, participants were taught how to draw a circle of a certain radius at the spot where the user taps the phone screen.

Participant Question: "Can we add a button or something to like change the radius?"

Teacher-initiated Discourse Moves (that influence the curriculum trajectory): The facilitator used this opportunity to explain the idea of *hard-coding* values versus being able to *dynamically* change a value based on *user input*. This led to a program enhancement where the idea of a "text box" for user input was introduced and the radius was set based on the value specified by the user.

Analysis: Crucial but oft-used mechanics of programming were introduced based on a perceived need in the app detected by a learner.

4.1.4 Illustrative Example #4

Scenario: The same 'Paint' app described above originally had only 4 buttons – one for each color red, blue, green, yellow. The participants were playing around with the app-

Participant Question: "Is there an Erase button to bring the screen back to a clean canvas?"

Teacher-initiated Discourse Moves (that influence the curriculum trajectory): This, again, was a design invitation of a critical omission of an essential feature that gave an opportunity to discuss how to "reset" a system state, and also how a user requirement dictates new functionality.

Analysis: Once again, a commonplace but crucial computational concept was introduced based on a perceived need in the app identified by a learner.

4.1.5 Illustrative Example #5

Scenario: After incorporating the 'Erase' button in the same 'Paint' app described in the subsections above, as students were playing around, making funny images like smiley faces, one participant made a rather pretty drawing of a flower and asked the following question.

Participant Question: "What if we wanted to save these drawings?"

Teacher-initiated Discourse Moves (that influence the curriculum trajectory): This, again, was a design invitation of an omission in the functionality. It prompted the facilitator to talk about persistence of data beyond the life of the program, working memory, RAM vs. hard disk storage, and a brief explanation of databases, too, even though it was not possible to incorporate these into the program just then.

Analysis: Once again, a perceived need in the app detected by a learner, prompted the organic introduction of fairly advanced CS concepts that would not normally be discussed so early in an introductory CS/programming session.

4.2 How did App Inventor for Android fare as a tool for fostering discourse and computational learning?

App Inventor for Android is a visual programming environment for developing mobile apps for the Android mobile platform that uses blocks like the popular Scratch programming platform. It was first developed at Google Labs by a team led by MIT's Hal Abelson and since early 2012 has transitioned to, and been freely available from MIT's Center for Mobile Learning. Like other visual graphical programming environments, it is relatively easy to use and allows early experiences to focus on designing and creating, while avoiding issues of programming syntax. By allowing novices to build programs by snapping together graphical blocks that control the actions of different dynamic actors on a screen, AIA, like Scratch, quite literally makes programming a snap. Recently published literature has described its efficacy in introducing computer science to high school students and teachers as well as introductory CS courses for undergraduates [4, 11, 19, 25], especially those with little prior background in computer programming. Also previously described are lists of what works and areas of improvement for AIA [19].

Little has been written about AIA for introducing programming and CS concepts to children in middle school. In our experience, AIA was an easy-to-use tool that was also excellent in motivating our 11 to 14 year old participants with little to no prior experience in programming to not only learn to program but also enjoy their first real programming experience. Mobile phone apps, and games in particular, are familiar territory for all tweens and teens today regardless of gender. In past literature, robotics and video game creation have been cited as suitable settings for children to learn computational thinking skills [18]. However, these as not very democratic in that they appeal more to male interests [6]. Recently introduced computational kits for creativity like e-Textiles seek to address this, but end up swinging almost all the way in the other direction, and appeal largely to girls. With the benefit of first-hand experience in introducing children to all these "tangible" computational tools in the past, we find AIA to be the most gender neutral and truly "democratic" among them all. Recent efforts to engage girls in computing, especially since stark enrollment numbers were published five years ago [13], as well as recommendations for engaging girls through "computing in context" [2, 8], provide a compelling rationale for the use of a tool such as this.

AIA helped us meet the primary goal of the workshop – that of providing a friendly platform that would allow novice middle school programmers to create exciting and fun apps. The thrill of hearing the first "meow" upon tapping the image on the screen was evident in the squeals, multiple utterances of "this is so cool" from around the table, as well as repetitive animal sounds as well as text-to-speech audio that filled the morning session as students repeatedly and excitedly tapped the screen or shook the phone. The complexity of the computational constructs that was demonstrated in the final projects by the end of the day was evidence of the completeness of the AIA feature set as a tool to expose novices to most all of the foundational elements of CS and CT that introductory computing curricula strive to cover.

Lastly, and perhaps most importantly (given the pedagogy employed in this curriculum), was the AIA's ability to foster natural engagement with the mechanics of programming and computational thinking through discussions and questions. Past studies on the language and structure that children and adults naturally use in solving problems before they have been exposed to programming show that "an event-based or rule-based structure" was often used, where actions were taken in response to events. "When PacMan loses all his lives, it's game over" would be an example of how people would generally describe their thinking process [12]. AIA's event-driven architecture for creating apps that responds to user or system-generated events was thus in keeping with the way novices approach problem-solving through programming; and was conducive for fostering organic discussions of how an app would be designed to achieve a desired result in response to user actions. Additionally, the complexity of event-driven programs the students were able to create with AIA in the course of a one day workshop went far beyond the limited "if-sensor-then-start/stop-motor (or LED)" type of programming that typically dominates the 'low floor', but not necessarily 'high ceiling' computational experiences in robotics and e-Textiles [3].

5. CONCLUSIONS

Despite the vast body of research on children and programming in the 1980s and the increased attention being given to computing education in K-12 today, educators are yet to establish the best pedagogies for introducing children to programming and computational concepts. It is thus worthwhile to bring research in the learning sciences to bear on the design of computing education curricula for younger children, and explore the value of the "social" aspect of the environment in the learning experience. This paper describes how talk, specifically, when used consciously and productively in an introductory CS curriculum for young learners can shape the ***process of development*** of CT. While data from other sources captured during this workshop needs to be analyzed to more closely investigate the impact of discourse on the development of specific elements of computational thinking, preliminary evidence from this exploratory study on the role of "computational discourse" for developing these competencies is very encouraging, and merits deeper inquiry as well as more widespread usage in curricula, especially for younger learners of CS and programming. Also, this opens the door to other possibilities in learning contexts as well. Through examining talk and interactions, targets of student difficulty could also be identified along with discursive strategies to deal with them.

Capturing teachable moments and facilitating additional learning for children is not a unique pedagogical practice; this approach is in fact fairly standard in good teaching for a variety of subjects. However, anecdotal evidence as well as a review of academic literature suggests that it is not usually employed in introductory programming or CS classrooms in K-12 (or even at the undergraduate level). Evidence from this workshop suggests that computationally rich environments may be consciously ***designed for computational discourse*** to help children develop a vocabulary that is faithful to CS as a discipline, as well as an understanding of the fundamentals of programming and computational thinking concepts and skills in a structured social setting inside or outside the traditional classroom.

As far as tools for computational thinking are concerned, App Inventor for Android appears to be emerging as a strong candidate-programming environment for use with early learners, especially those in the tween/teen age group. It stacks up well against current popular choices such as Scratch and Alice, and in some aspects appears to be an improvement on them by allowing for creative app building—something all teens, including girls, are eager and motivated to do—while still engaging with complex CT concepts including procedural and data abstraction, iterative and recursive thinking, structured task breakdown, conditional and logical thinking, and debugging.

To summarize, while this experience report is limited by a small sample size, and as such its findings are not generalizable, it is illustrative and provides a foundation—and direction—for much needed further work in the area of studying the development of computational competencies in school-age children including pedagogies and tools that support such efforts and appropriate curricula to achieve that goal.

6. ACKNOWLEDGMENTS

Our thanks to Prof. David Wolber at University of San Francisco for use of USF's HTC Android phones for this workshop, Prof. Paulo Blikstein for the use of the Transformative Learning Technologies Lab at the Stanford School of Education for conducting the workshop, audio/video recording equipment and laptops for the workshop participants. Thanks to the National Science Foundation (NSF) support of the LIFE Center (NSF-0835854) contributing to this work. Thanks also to the Stanford University School of Education for lending laptops for the workshop as well as providing funding for software, refreshments and honorariums for workshop volunteers.

7. REFERENCES

[1] Cazden, C.B. 2001. *Classroom discourse: The language of teaching and learning (2nd ed.)*. Portsmouth, NH: Heinemann.

[2] Cooper, S., and Cunningham, S. 2010. Teaching computer science in context. *ACM Inroads.* 1,1, 5–8.

[3] Eisenberg, M., Elumeze, N., MacFerrin, M., and Buechley, L. 2009. Children's programming, reconsidered: settings, stuff, and surfaces. In *Proceedings of the 8th International Conference on Interaction Design and Children, Como, Italy*.

[4] Gray, J., Abelson, A., Wolber, D., and Friend, M. 2012. Teaching CS principles with app inventor. In *Proceedings of the 50th Annual Southeast Regional Conference* (ACM-SE '12). ACM, New York, NY, USA, 405-406

[5] Grover, S., and Pea, R. 2013. Computational Thinking in K–12: A Review of the State of the Field. *Educational Researcher.* To be published.

[6] Kafai, Y.B., Peppler, K.A., Burke, Q., Moore, M., and Glosson, D. 2010. Fröbel□s Forgotten Gift: Textile Construction Kits as Pathways into Play, Design and Computation. In *Proceedings of the 9th International Conference on Interaction Design and Children. Barcelona, Spain: ACM,* 214-217.

[7] Kurland, D.M, Pea, R.D., Clement, C., and Mawby, R. 1986. A study of the development of programming ability and thinking skills in high school students. *Journal of Educational Computing Research*. 2, 429-458.

[8] Margolis, J., & Fisher, A. 2002. *Unlocking the clubhouse: Women in computing.* Cambridge, MA: MIT Press.

[9] Michaels, S., Sohmer, R. E., and O'Connor, M. C. 2004. Classroom discourse. In H. Ammon, N. Dittmar, K. Mattheier, & P. Trudgill (Eds.), *Sociolinguistics: An international handbook of the science of language and society* (2nd ed., pp. 2351–2366). New York: Walter de Gruyter.

[10] Michaels, S., O'Connor, C., and Resnick, L. B. 2008. Deliberative discourse idealized and realized: Accountable

talk in the classroom and in civic life. *Studies in Philosophy and Education*. 27,4, 283–297.

[11] Morelli, R., de Lanerolle, T., Lake, P., Limardo, N., Tamotsu, E., and Uche, C. 2011. Can Android App Inventor Bring Computational Thinking to K-12? *Proc. 42nd ACM technical symposium on Computer science education (SIGCSE'11)*.

[12] Myers, B. A., Pane, J. F. and Ko, A. 2004. Natural Programming Languages and Environments. *Comm. Of the ACM*, (Sept. 2004), 47-52.

[13] NCWIT. National Center for Women & Information Technology. (2007). NCWIT Scorecard 2007: A report on the status of women in information technology. Boulder, CO: National Center for Women & Information Technology. Retrieved September 15, 2008, from http://www.ncwit.org/pdf/2007_Scorecard_Web.pdf.

[14] National Research Council. 2011. Committee for the Workshops on Computational Thinking: *Report of a Workshop of Pedagogical Aspects of Computational Thinking*. Washington, DC: National Academies Press.

[15] National Research Council. 2012. *A framework for K-12 science education: Practices, cross-cutting concepts, and core ideas*. Washington DC: National Academy Press.

[16] Papert, S. 1980. *Mindstorms: children, computers, and powerful ideas*. New York: Basic Books.

[17] Papert, S. 1991. Situating Constructionism. In S. Papert & I. Harel (Eds.), *Constructionism*. Cambridge, MA: MIT Press.

[18] Repenning, A., Webb, D., and Ioannidou, A. 2010. Scalable game design and the development of a checklist for getting computational thinking into public schools. In *Proceedings of the 41st ACM technical symposium on Computer science education (SIGCSE '10),* 265–269.

[19] Roy, K. 2012. App inventor for android: report from a summer camp. In *Proceedings of the 43rd ACM technical symposium on Computer Science Education* (SIGCSE '12). ACM, New York, NY, USA, 283-288.

[20] Sohmer, R., Michaels, S., O'Connor, M.C., and Resnick, L.B. 2009. Guided construction of knowledge in the classroom: The troika of talk, tasks and tools. In B. Schwarz, T. Dreyfus & R. Hershkowitz (Eds.), *Transformation of knowledge through classroom interaction* (pp. 105-129). Abingdon, UK: Routledge.

[21] Wilson, C., Sudol, L.A., Stephenson, C., and Stehlik, M. 2010. *Running on Empty: The Failure to Teach K-12 Computer Science in the Digital Age*. New York, NY: The Association for Computing Machinery and the Computer Science Teachers Association.

[22] Wing, J. 2006. Computational Thinking. *Communications of the ACM*. 49(3), 33-36.

[23] Wing, J. 2011. Research Notebook: Computational Thinking—What and Why? *The Link Magazine, Spring 2011*. Carnegie Mellon University, Pittsburgh. Retrieved September 7, 2012 from http://link.cs.cmu.edu/article.php?a=600

[24] Wolber, D., Abelson, H., Spertus, E. and Looney, L. 2011. *App inventor: create your own android apps. Sebastopol, CA: O'Reilly Media.*

[25] Wolber, D. 2011. App inventor and real-world motivation. In *Proceedings of the 42nd ACM technical symposium on Computer science education* (SIGCSE '11). ACM, New York, NY, USA, 601-606.