Tangible Media Approaches to Introductory Computer Science

Evan Barba
Georgetown University
Washington, DC
evan.barba@georgetown.edu

Stevie Chancellor
Georgia Institute of Technology
Atlanta, GA
schancellor3@gatech.edu

ABSTRACT
Computing is an increasingly important component of many jobs and demand for computing skills is far outpacing the number of computationally literate workers available. Non-majors, adult learners, and other non-traditional students can potentially fill some of these positions. However, traditional CS education pathways do not currently address the unique needs of these students. New approaches to CS education that fit with the goals and lifestyles of non-traditional CS students are needed. In line with Computational Media approaches known to be successful with non-majors, we designed and implemented two graduate-level courses, one using a Pixelsense and the other using Arduino, to teach computational thinking, programming, and design skills. We compare findings from these two courses with specific focus on non-major graduate students, but including topics relevant for traditional CS educators, such as, the importance of choice of platform, structure of assignments, maintaining student motivation, and the impact of self-guided final projects.

Categories and Subject Descriptors
K.3.2 [Computer and Information Science Education]

General Terms
Design, Experimentation, Human Factors, Management

Keywords
Tangible Media, Non-majors, Computational Media, Late Bloomers, Physical Computing, Introductory Computer Science

1. INTRODUCTION
The need for computer science (CS) literacy in the workforce has grown rapidly and does not show any signs of abating. The U.S. Bureau of Labor Statistics projects that, by the year 2020, half of all Science, Technology, Engineering and Math (STEM) jobs will be in computing-related fields. Many of these new job opportunities incorporate computing into the work practices of previously unrelated fields. Journalists, public relations analysts, lawyers, advertisers, educators, and many others are expected to create computational artifacts and use computational tools for data analysis, information visualization, simulation, and more. Students in these disciplines are looking for introductory computing courses that authentically reflect the way computing is being adopted in those professions [3]. Moreover, workers already in those fields are finding that they need introductory computing courses to advance in their careers. These non-major and non-traditional "late blooming" CS students are a growing demographic in CS education that warrants our attention.

We describe two experimental approaches to teaching concepts in computer hardware, software engineering, and interaction design to graduate and undergraduate students in an interdisciplinary educational context. We designed and taught two courses. The first was "Tangible and Embodied Computing" (TEC), a course that introduces computer programming topics and user interface design in the context of a Microsoft Surface v1 (Pixelsense) tabletop computer. The second was "Interaction Design Electronics and Semantics" (IDEAS), a course using a physical computing approach that included Arduino and electronics in addition to programming and design instruction. We consider both of these courses to fall in the category of "Tangible Media" [6,7]. Our courses represent the extremes of Tangible Media. The experiences in TEC couple everyday non-computational objects to processes and entities in the virtual world and therefore heavily emphasize the virtual. IDEAS, in contrast, emphasized the physical by embedding digital sensors and actuators into physical artifacts. Both TEC and IDEAS used a constructionist learning approach in which students combined physical objects and software to create interactive artifacts. We compare the effectiveness of these approaches for learning basic concepts in computational thinking for a small sample population.

We found that students were able to acquire programming and design skills and responded favorably to each course. Students in both courses also demonstrated soft skills associated with communication and problem solving. Each course had its own benefits and drawbacks for students. TEC was more difficult with its expectation of coding and programming, but students reported that they had more insight into computing as a professional practice. IDEAS was received better by the students but provided students less marketable skills despite learning some of the same core principles of computational thinking. Our discussion includes insights into student motivations, different learning styles, the effectiveness of tutorials, and teaching "code as a component" of larger multifaceted project. In addition to findings that are generally relevant to introductory CS courses, we identified a number of factors that make the population of adult learners we observed in our studies unique and suggest areas of future research relevant to this demographic.

ITICSE '15, July 04 - 08, 2015, Vilnius, Lithuania Copyright is held by the owner/author(s). Publication rights licensed to ACM. ACM 978-1-4503-3440-2/15/07 $15.00
http://dx.doi.org/10.1145/2729094.2742612

2. BACKGROUND AND RELATED WORK

The majority of research in introductory CS courses focuses on undergraduate education and address issues of pedagogy, language selection, and course assignments [11]. The target of most of these courses is undergraduate CS majors, students who are expected to continue building the foundations of computational literacy well into their undergraduate careers. Notable exceptions are courses created explicitly for non-majors that take a "Media Computation" approach to introductory CS [2]. That research shows that introducing computing as new media is effective for non-majors and promotes more engagement and learning. We hypothesized that adult learners might also benefit from a Media Computation approach that is specifically focused on Tangible Media. This focus on Tangible Media as a subset of Computational Media came directly from polls of student interest conducted by our department, which showed that learning technologies associated with "Making" and the "Internet of Things" were highly desired by our incoming students.

The interdisciplinary graduate program where our courses were taught is one example of a growing number of programs that offer students the opportunity to do interdisciplinary graduate work around the production and consumption of technology. Our program caters to students with backgrounds in the humanities and social sciences that want to integrate the use of information communications technologies into their existing skillsets. Typically this is done to improve their career outlook by making them more competitive applicants for jobs and doctoral programs. Most of our students begin with little to no experience in programming but are sophisticated and critical consumers of digital technologies. Although they are more highly educated and often have more work experience than undergraduate students, they are similar to undergraduate non-majors in terms of their level of hands-on experience in CS. Classes in this department are small (enrollment is capped at 15 students) and the department often attracts advanced undergraduate students from other areas. For these reasons we chose to base our course designs in a Media Computation approach that leveraged students' expressed interest in working with Tangible Media.

Tangible Media as an introductory approach to CS has been discussed before. However, these efforts either focus on primary and secondary CS education [4], on informal settings [8], or on learning topics outside of CS [5,10]. The most typical approach is one in which computer code is physically embodied in some modular object and then combined and recombined to create different behaviors. This is an elegant and effective approach for elementary school and young adults, although some evidence suggests that they can introduce problems around knowledge transfer to more traditional domains [1,9]. The graduate students we work with have demonstrated the capacity to learn more sophisticated techniques and are specifically asking for skills and knowledge that is directly transferrable to workforce skills; therefore, more simplistic approaches that reduce the complexity of computational concepts are not appropriate for our context. Still, there is much that is applicable. Namely, that students' intuitive understanding of the physical world is an effective gateway to a more sophisticated understanding of the digital virtual world. This supposition proved true in both TEC and IDEAS.

Some of the key concepts of the Computational Media approach noted in [2] are listed below, summarized in our own language:

- authenticity and relevance: how closely the classroom experience matches real world practices
- creativity: how well the class supports the creation of personally expressive and meaningful artifacts
- tailoring: how closely the course matches individual student needs and wants
- motivation: whatever it is that keeps students engaged for the duration of the course

We hypothesized that these broad concepts would also apply to our students, and we designed our specific learning objectives (bolded below) for each of these courses to reflect these concepts. We forced students to **learn to ask effective questions** with authentic sources of information like classmates, instructors, online forums, or search engines. We gave students **relevant and authentic technology** in the Arduino and Pixelsense. Both syllabi emphasized the importance of **learning generalizable skills and concepts** that could extend to other domains. We gave them the opportunity to be creative by designing their own projects and insisted that they **become comfortable with an iterative design methodology** that is commonly employed in software engineering and other design-related work. Students filled out a pre-class survey (TEC) or a blog post (IDEAS) detailing prior experience with design and programming for purposes of later assessment, but which also asked them to provide their own goals for the class and we referred to and assessed these goals frequently during our one-to-one interactions with individual students. By fostering an attitude of ownership over their projects, we hoped that students would remain motivated to learn essential concepts and practices in CS that could transfer to their career goals later.

The courses were taught by the same instructor and teaching assistant (instructors hereafter) and combined focused tutorials and homework assignments with longer, student-driven projects. Tutorials were started in-class under supervision and were finished after class as homework. Students proposed ideas for their final projects and the instructors helped scale and develop timelines for each. Those who demonstrated a better grasp of the material in their tutorials were encouraged to develop more robust final projects. Evaluation of students' performance in both classes was also done similarly. Smaller programming and design projects came with a list of criteria needed to receive high marks. Final projects were demonstrated to the class. We gathered data about student experiences in both courses for purposes of formative assessment. Learning progress toward the stated goals was evaluated through course blog posts, interactions with the instructors, recorded conversations, and grading of the projects. Students were required to submit documentation alongside their programs. TEC also had a post-course focus group conducted by an independent external evaluator.

3. COURSE OVERVIEWS AND RESULTS

3.1 TEC

TEC was taught in Fall 2013 to eight students, one undergraduate and seven graduate students, five men and three women. Only the undergraduate had any previous programming experience. The structure for TEC consisted of six short tutorials to guide the students through programming tasks specific to the Pixelsense environment, three homework assignments that extended these tutorials to more complex problems including a four-function calculator, and a final project defined by the student and approved by the instructor.

3.1.1 Pixelsense Platform

There were several reasons we chose to use the Pixelsense table. First, it is an "authentic" real-world device that is not a sandbox for beginners. Furthermore, the touch and gesture recognition

capabilities are now familiar to most users and common across mobile and tablet platforms, while the large size, tabletop orientation, and object tracking capabilities are unique enough to be cutting-edge. The biggest downside to the Pixelsense table is its lack of online developer resources. We did not anticipate this in our design phase because of the availability of Microsoft's own documentation and community forums; unfortunately, these were often incomplete and not appropriate for beginners. To compensate, we created a course blog as a repository of knowledge for our class. Although students used the blog to ask a few questions, instructors almost always gave the answers.

3.1.2 Assignments

TEC's tutorials included topics in basic programming (syntax, control flow, etc.), object-oriented and event-driven design principles (encapsulation, polymorphism, event listening), user interface design (affordances, visual hierarchies), and API and library management. The first three tutorials focused on XAML and the remaining tutorials focused on C#. Each tutorial took an incremental approach with detailed descriptions of programming concepts, screenshots of expected output, and code snippets. We designed the progression so that students would learn how to create simple interface elements through XAML and carry this knowledge with them into the development of C# "backcode."

Students completed tutorials in class after a 45-60 minute lecture on the topic, giving them around 90 minutes of class time for each. Unfinished tutorials were assigned as homework to allow slower students to catch up, and faster students often used this time to add flourishes to their projects, suggesting that they were engaged and motivated with the work. For example, some students added additional components and functions to the calculator project and added design and interaction components (like 3D printed objects) to their final projects.

The similarity of XAML to HTML and XML make it both familiar to beginning students and relevant beyond the class. Nearly all the students were able to complete the first two tutorials in class, and even those who struggled were able to complete them as homework with some assistance given during office hours. However, the early burst of confidence in programming in XAML lulled students into a false sense of security; specifically, students believed that C# would be as simple or as quick to write as XAML. One student described the transition between XAML and C# as "a brick wall a month in."

The use of the common and powerful C# programming language was clearly relevant outside the course and students were excited to know that what they were learning could be applied across any Microsoft platform. But the relationship between XAML and C# also proved to be a major stumbling block. The concept of "binding" interface elements to C# was troublesome for all students. Students had difficulty abstracting behavior away from the visible interface elements and into C# program logic. As one student noted, "I felt like everything that could do something should be on the screen." Despite integrating the two languages incrementally in the tutorials and the homeworks, most students initially could not see how the two were related. Although most students were eventually able to grasp this abstraction, a few did not, and this was a continued source of frustration as they tried to move forward.

Each student was responsible for two short homework projects that directly extended the skills taught in the tutorial by asking them to reimplement the design patterns used in the tutorial in a different context. The majority (5 of 8) of the students completed these with few problems. However, most had a superficial understanding of the concepts and techniques involved and when they were required to transfer these concepts onto different programming elements (from a button in the tutorial to a slider in the homework, for example) some students struggled despite having successfully completed the tutorial. We came to understand that this was because students were unable to see the larger pattern. They were daunted by the idea that a slider behaved differently than a button, even though both are implemented in the exact same way (only a change of name and one additional variable was needed).

The last homework was a small project designed to integrate the six tutorials and related concepts together by programming a digital four-function calculator. This is where we saw a clear divergence in student aptitude. Some completed the calculator in a timely matter, some completed it with sloppy results and errors, and two students were unable to complete it for many weeks. Once the class moved on to final projects, these students had to split their attention and became overwhelmed with the amount of work left for them. One student asked for an incomplete in the course so that they could finish the work over the next semester.

Another interesting finding was how different aspects of programming confused students. For some, syntax was difficult but overall programming structure, including object-oriented concepts and event handling, were intuitive. Still others were fine with syntax, control flow, and structure but struggled with softer skills such how to iterate and improve code incrementally or ask effective questions. Success in one of these skills could not be predicted by early success with homework or the midterm project. Students who submitted homework assignments early and showed strong aptitude for programming syntax and language were just as likely to struggle later in the semester as lagging students. One student completed all of the homeworks rapidly and even became a resource for others but could not decompose their final project into manageable pieces and struggled for the latter half of the semester. These individual differences are often hidden in larger classes where everyone proceeds in lock step. Identifying these individual differences in learning specific topics and developing tailored techniques match is an area we have identified as having tremendous value to CS education at all levels.

In an attempt to let students manage their workload with their own schedules and give students agency in their learning, we allowed students to work at their own pace while completing assignments. Deadlines were strongly suggested, but flexible. The majority of our students were graduate students, and we expected them to have the maturity to manage their time. This approach met with mixed results. Students that were engaged made appropriate progress on their homework and mini-projects and submitted assignments at the recommended deadlines. These students received quick feedback on their work that they could then apply to future work. However, struggling students could put off the discomfort of working through difficult concepts until it was far too late for instructor feedback to be incorporated into their work. Rather than creating a relaxed atmosphere for cooperative work between students with different skillsets, this approach allowed students to diverge onto their own paths and rarely interact students outside their skill level.

3.1.3 Final Projects

In the last eight weeks, the pedagogical style of the course shifted to a studio model to give students the time and support needed to work on their final projects. Final projects included a "whack-a-

mole" game, a fantasy football draft application, and an application integrating geo-tagged Twitter posts with a map and physical objects. TEC students reported that they enjoyed the process of planning, designing, and coding the final project. One student said, "Getting 'sent in' to just accomplish our projects and do it ourselves - it's been very hands on - this is DEFINITELY the way to do this!" Another said, "After this class, although it was super tough, I feel more confident in my abilities to accomplish things, especially in technology." These statements suggest that what was learned was not programming content but softer skills and a deeper understanding of what is involved in CS.

Overall, five of the eight students were able to produce completed final projects. Of those five, three were at a level that would be expected of high-quality introductory CS students. However, these successes were hard-fought and took a significant toll on the instructors as well as the students. Each student's project was individually defined and scoped and each student had their own distinctive areas of difficulty. The emerging instructional style was a continuous loop of creating ad-hoc individualized scaffolds appropriate for each student. As students reached an unsolvable problem, they would approach one of the instructors who would need to context shift into the project, identify the reason for the difficulty, and devise and explain a solution for the student. Sometimes this was straightforward, like incorrect syntax or scope, and other times it required real debugging that a novice programmer could not be expected to perform on their own. Even when students tried to find solutions online to their problems, results were so scattered for the Pixelsense table that students could not proceed. By the time every project had been troubleshot, the students would present a new round of problems. This close mentoring was exhausting both for instructor and student and would be difficult to scale to larger class sizes. Furthermore, it privileged making progress toward the final goal over a student's self-discovery of a solution, and possibly hampered learning.

3.2 IDEAS

IDEAS was offered in Spring 2014 to fifteen students, two undergraduate and thirteen graduate, seven women and eight men. Three students from TEC enrolled in IDEAS. None of the students in IDEAS had experience with electronics or prototyping but four had previous programming experience. With nearly twice the enrollment, it was obvious from the beginning that something about IDEAS was more attractive to this population of students.

3.2.1 Assignments

IDEAS took a physical computing approach that taught both electronics and programming through two progressions of tutorials and projects. As with TEC, students completed tutorials in class after a 45-60 minute lecture and unfinished tutorials were completed as homeworks. The first progression of three tutorials reinforced the basics of electronics and prototyping. Topics included how to use a multimeter to test components, how to draw a circuit diagram, and how to assemble parts on a breadboard. Students found these straightforward and interesting and were engaged throughout, but also showed great apprehension about working with electricity. Most of the students completed these tutorials easily.

The first group of tutorials culminated in a longer project where the students assembled an Atari Punk Console (APC).[1] The APC is a beginner electronics project that produces sounds similar to an Atari 2600 gaming system. Students were also required to prototype an external housing for their APC that made it a suitable intervention for a particular task or environment. Completed projects included a series of flex resistors in a box to make the APC act as a windchime, adding a thermistor to make a boiling water alarm, and attaching the APC to the back of cardboard character for use at sporting events.

Despite having successfully completed the tutorials and reporting comprehension of the lecture material, almost every student had tremendous difficulty with the APC project. This is because students could not comprehend the wiring diagrams. While the tutorials had them assemble components in a stepwise manner with diagrams, the jump to a more complex diagram and lack of incremental instructions increased the difficulty too quickly. Although students could eventually assemble the APC it took twice as long as expected. They did, however, do excellent design work, creating interesting and functional external housings using a range of materials.

Whereas the lack of tutorials for the Pixelsense required us to create them from scratch for TEC, electronics tutorials are plentiful and we hoped that having a variety of explanations and perspectives would aid in student learning. We directed students to tutorials from Instructables[2] and similar sites for the initial electronics projects and the tutorials supplied on the Arduino website for six Arduino programming tutorials, but students were encouraged to use whatever they could find on their own. Every student was given an Arduino Uno prototyping platform to use for the class, and we supplied them with additional electronic components as needed. Despite these tutorials being more thoroughly vetted than the ones we created in TEC, students had similar difficulty seeing the broader patterns being introduced. Although they did get a sense of what the Arduino was capable of, and exposure to programming concepts like variables and loops, they also showed little ability to transfer this knowledge into new contexts. We had to continually refer them back to relevant tutorials and in many cases had to explain why it was relevant to the current situation.

After completing the tutorials, students defined a final project in consultation with the instructors. Students could work in teams to develop a new project or working individually to iterate on their APC project. Our choice to allow groups to form was a direct consequence of our experience in TEC, where we ended up tutoring students on individual projects. IDEAS had more students; requiring individual projects would have made the same level of assistance impossible. IDEAS had the same number of projects as TEC, but the workload was less taxing on the instructors and less frustrating for the students. This was partly a reflection of the number of people working on a project but was also due to the amount and quality of the online assistance. Students had many examples to draw from and it was likely that someone had already encountered problems similar to their own and posted solutions on an online forum. In the end, the students completed projects of equal complexity with far less assistance.

The availability of online sources introduced one complication that we did not foresee, but also provided an authentic learning opportunity. By relying on tutorials, we implicitly suggested that following a tutorial was an acceptable final project rather than creating something new and innovative. Three student groups

[1] http://en.wikipedia.org/wiki/Atari_Punk_Console

[2] http://www.instructables.com

took this approach, and we allowed it because creativity and originality were not stated as grading criteria. Students who chose this path still had difficulties and it led to some interesting observations. Parts do not always match completely, code is not always updated and maintained, pictures and descriptions are not always accurate. Although they were not working on projects they defined themselves they encountered the same problems that teams working on original projects did. Interestingly, they did not appear any less motivated. This suggests that motivation is not necessarily tied to creativity and ownership over a project and the idea of "measuring up" is a powerful motivator as well. These students were motivated by not wanting to feel a sense of inadequacy. The only real problem with this approach was that students ignored solutions that deviated from the prescribed steps. One student tried to install a software MIDI converter used in the project tutorial and became so preoccupied with making it work that they didn't think to try another converter until the instructors urged them do so. Students who reproduced projects they saw online still learned how to troubleshoot electronics and debug code. In fact, because they did not start from scratch with their own idea they learned a valuable and authentic skill. Many jobs require new employees to enter a project that has been ongoing for years or to adapt existing work to suit new purposes.

Students who designed their own artifacts went through the entire design process and we believe generally learned more in the course regardless of the quality of their final projects. One group's project integrated pressure sensors into a seat cushion to alert sitters when they had been sitting for too long by playing an alarm. Although this product exists already, the students prototyped everything themselves in multiple iterations, found relevant code snippets and additional hardware, and were able to integrate and patch together the code to produce the logic they wanted.

In general students in IDEAS remained motivated throughout the semester, largely because they felt the projects were producing polished artifacts that they could show their friends. They felt great pride in showing their creativity. Gaining social capital through presenting polished artifacts proved to be a powerful motivator.

4. COMPARISON

IDEAS was the more popular course from the start. Nearly twice the number of students in TEC enrolled in IDEAS. Students cited various reasons why they were more interested in the class. The perceived accessibility of the subject matter, "design" versus "computing," appealed to more students even though students learn design skills and computing skills in both classes. Students also reported that IDEAS seemed like an easier class. Several of the students in IDEAS told us they had considered taking TEC, but the topic seemed too hard and out of reach for them. With its focus on building artifacts, IDEAS was thought to be easier because it appeared more like "arts and crafts."

Although they were aware that C# is a widely used programming language, TEC students' initial enthusiasm waned when they realized it would take more than a single class to become proficient. Combined with the physical immobility of the Pixelsense table, students were left feeling like the technologies involved were specialized and inaccessible, a feeling that was reinforced by the lack of prominent online resources. In contrast, there are sites that offer technical assistance and discussion of electronics projects. These projects appear in places that are not dedicated solely to these technologies (Instructables covers topics outside of electronics), so these skills appeared more broadly supported and relevant outside a niche audience. The design and prototyping skills in the IDEAS curriculum are attractive to students because they are trending as part of a national discussion surrounding "innovation" and "Making," and students felt like they were learning things relevant to possible future careers that were more creative and enjoyable. The skills learned in TEC were perceived as more constrained and device-specific despite being more relevant to the workforce.

Still, despite the perceived inflexibility of the TEC coursework, students also recognized that they had an authentic programming experience, learned to become conversant with programmers, and got a better understanding of, and respect for, professional programmers. Increasing general awareness of how authentic computing is done in the real world was actually quite a useful skill, as many of these students were expected to communicate with more technical personnel in their jobs. One student told this story, "At work the other day, one of the designers, I saw them working with code and programs, and I thought - I know what you're doing! I could look at what you're working on, and probably understand it!" This is the essence of computational literacy.

4.1.1 Code as a Component

One of the major differences between the technologies used in TEC and IDEAS is the role of code. In TEC, code is central. The end product is a computer program that includes physical components that have specific meanings in the context of the program. One student used playing cards, another used spoons, and another created a 3D print of targeting reticle as an interface. These helped bridge the divide between physical and virtual; however the projects were still perceived as "computing projects," and, to their dismay, the students felt more like programmers than designers.

In IDEAS this was largely reversed. Students' projects ended up as physically distinct forms with unique characteristics and the code was hidden away as one of the components that make the object behave. This helped take some of the pressure off of the students to learn programming. Objects could behave somewhat erratically as they debugged the code but still appeared "finished" because they had a physical manifestation. Students could take them home, talk about them, and point to the various components as evidence of their work and ingenuity even if it didn't function. When Pixelsense programs crashed, students were dismayed that the overall impression was of a non-working artifact with little visible evidence of their efforts.

The major advantage to IDEAS, however, was in students' ability to reason about how to achieve the behaviors they wanted. Students were better at identifying the logic of physical interactions than the logic of their more abstract tabletop applications. For example, the alerting seat cushion required integration with additional software for data-logging, timing, and audio playback, which were more complex than many of the tabletop applications created in TEC. These students independently arrived at the notion that their artifact needed to have different "states" to handle the different combinations of sensor input and audio output. The logical flow of the program followed the logical flow of their physical interaction and they could repeatedly perform these actions in the physical world to think through the different conditions that would arise in the use of their artifact. There was real learning of computational thinking, abstraction specifically, that is arguably more valuable than understanding the intricacies of C#.

4.1.2 Soft Skills

The most relevant soft skill that students learned was how to properly ask for help, from both the instructors and from online sources. There was marked improvement throughout both courses in the vocabularies students used to describe the problems they encountered. These improvements were more pronounced in TEC. Programmers build up a sense of how to ask a good question and identify a relevant answer over time. Even knowing how to properly search to get relevant results took considerable time to master. At first, students would use vague and unspecific language. For example, referring to variables as "numbers" instead of "ints" or "floats." As they realized these differences carried meanings, they paid more attention to their language and would use the more specific name. Developing these softer skills are a part of CS literacy that is often overlooked and is particularly salient for students in this demographic who are often asked to be liaisons between programmers and clients.

4.2 Learning from Tutorials

We have already mentioned the similar results in the two classes regarding the effectiveness of tutorials, and we find this point indicative of a larger problem that needs more study. Our observations around the effectiveness of tutorials have led us to suggest the following: tutorials are best treated as a kind of scaffolding rather than a primary source and, when used, should not be scaffolded any further. This untested hypothesis needs deeper investigation, but it raises an issue of critical importance to both online and classroom settings beyond CS education alone.

When students develop projects on their own, there is no known pre-existing solution. This forces them to synthesize information into a solution of their own devising. Scaffolding this process of design with multiple tutorials that reveal parts of the solution is a useful pedagogical tool in this case. Students are still learning to identify problems, search out, sort, and map possible solutions, and integrate knowledge. Tutorials serve to support learning specific techniques when students have independently decided they need that support. Projects in which students had an original idea, and assembled a new artifact without a roadmap often used tutorials this way and were the most successful. In these cases students were able to decompose a problem into sub-problems and locate tutorials on how to solve these sub-problems. The student who made a boiling water alarm first followed tutorials on how to connect thermistors to the Arduino. She then had to program it to respond at the correct temperature ranges by assembling code from different tutorials, such as those on reading the Analog-to-Digital Converter and playing tones. These were sub-problems that she identified and solved herself.

Contrarily, when tutorials presented a stepwise recipe that defined both problem and solution, it was difficult for students to generalize these to other situations. Solutions to problems and questions encountered during the completion of the tutorials *should not be scaffolded*. Students must learn to seek out these answers themselves in order to transfer this knowledge to other contexts. This was a major problem that both courses encountered. Students would ask the instructor for "clarification" of what the problematic step was and to explain the concept they were missing. This process resulted in a completed tutorial, but little actual learning. The answers came too easily. Although many of the tutorials were also accompanied by an exercise that asked students to stretch and apply their knowledge, they were not effective at engraining any knowledge.

Tutorials seem to be most effective at showing students *how* to do something when they are already aware of *why* they need to do it, but the important step of being able to identify when a technique shown in a tutorial could be applied was crucially missing..

5. CONCLUSIONS

Through our work designing and implementing TEC and IDEAS, we found that students respond favorably to Tangible Media as computational and pedagogical tools. Many of our findings align with what is known about Media Computation approaches more generally. For CS1 instructors looking to engage their students, we think several of the themes from these courses could be implemented into undergraduate classes. Finally, two questions, "How do we use tutorials to facilitate knowledge transfer?" and "What are the most effective techniques for addressing the unique needs of late-blooming CS students?" come to the forefront as areas for future research.

6. REFERENCES

[1] Barba, E., Xu, Y., Maclntyre, B., and Tseng, T. Lessons from a class on handheld augmented reality game design. *Proceedings of the 4th international conference on foundations of digital games*. ACM Press (2009), 2–9.

[2] Guzdial, M.Exploring hypotheses about media computation. Proceedings of the ninth annual international ACM conference on International computing education research - ICER '13, ACM Press (2013), 19.

[3] Hewner, M. and Guzdial, M.Attitudes about computing in postsecondary graduates. *Proceeding of the fourth international workshop on Computing education research ICER 08 293*, 5527 (2008), 71–78.

[4] Horn, M.S. and Jacob, R.J.K.Tangible programming in the classroom with tern. *Proceedings of ACM CHI 2007 Conference on Human Factors in Computing Systems*, (2007), 1965–1970.

[5] Horn, M.S., Solovey, E.T., Crouser, R.J., and Jacob, R.J.K.Comparing the use of tangible and graphical programming languages for informal science education. *Proceedings of the 27th international conference on Human factors in computing systems CHI 09 32*, (2009), 975.

[6] Hornecker, E. and Buur, J.Getting a Grip on Tangible Interaction : A Framework on Physical Space and Social Interaction. *Clavier*, (2006).

[7] Ishii, H.The tangible user interface and its evolution. *Communications of the ACM 51*, 6 (2008), 32–36.

[8] Lovell, E. and Buechley, L.An e-sewing tutorial for DIY learning. *Proceedings of the 9th International Conference on Interaction Design and Children - IDC '10*, ACM Press (2010), 230.

[9] Owen, S. and Disalvo, B.Graphical Qualities of Educational Technology. (2014), 14–17.

[10] Parkes, A.J., Raffle, H.S., and Ishii, H.Topobo in the wild: longitudinal evaluations of educators appropriating a tangible interface. *Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, ACM (2008), 1129–1138.

[11] Pears, a, Seidman, S., Malmi, L., et al.A survey of literature on the teaching of introductory programming. *SIGCSE Bulletin 39*, 4 (2007), 204–223.