Assessing the Impact of using Robots in Education, or: How We Learned to Stop Worrying and Love the Chaos

Douglas Blank and Deepak Kumar

Bryn Mawr College
Computer Science Department
101 North Merion Ave
Bryn Mawr, PA 19010 USA
{dblank, dkumar}@cs.brynmawr.edu

Abstract

For the past several years, we have been using robots in our introductory computer science course. Although this has been challenging for many reasons, it has also been very rewarding on a number of fronts, both for the students and for us. However, in order for this to occur, we had to adapt to what we perceived as "chaotic code." In this paper we describe lessons learned by watching what the students do, where they have trouble, and what they enjoy. Further, we discuss what the implications of focusing on creativity has had on teaching and assessment.

Introduction

In the summer of 2006, we began an experiment to teach introductory computer science courses using a small, inexpensive personal robot (Blank, 2006). The project uses an off-the-shelf robot (the Scribbler from Parallax, Inc.), a Bluetooth wireless communication and camera add-on board designed by our colleagues at Georgia Tech, a freely available textbook, and a Python library called Myro[1]. All of these materials have been developed under the auspices of IPRE (Institute for personal Robots in Education) which continues to examine the role of personal robots in education. Since that time, over a thousand students have used these materials at colleges across the United States, and beyond (Kumar, et al. 2008).

At Bryn Mawr College, the materials have been integrated into the course CS110: Introduction to Computing. This is designed to be the first course on computing for all students irrespective of whether they intend to major in computer science. We deliberately do not offer a separate introductory computer science course for non-majors: all students interested in studying computing take the same introductory course (Blank and Kumar, 2002).

The personal robot prototype that we use in this course can be used to draw pictures with a pen that sits in the middle of the robot, play music through a small speaker, take pictures through its camera, and be interactively controlled over wireless (see Figure 1).

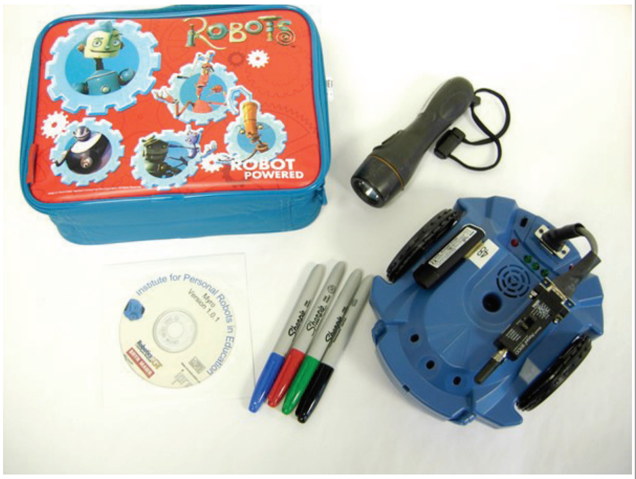

Figure 1: The Person Robot Kit that every student gets in the Introduction to Computing at Bryn Mawr College.

There are many technical hurdles in teaching and evaluating a course in which every student has their own robot. For example, we should have invested in stock from a battery manufacturing company when we first began the project. Also, we have had to adapt to the limitations of inexpensive robots that don't drive straight; however, this particular limitation can be seen as a virtue and embraced (Martin, 2008). In any event, these technical issues will no doubt be addressed and surmounted over time. But what

[1]You can find out more about our project at the Institute for Personal Robots in Education, http://roboteducation.org/

have we learned about the teaching of computer science more broadly?

In assessing our results so far, we find it useful to reflect on the changes that we have made in our assignments, lectures, and, more importantly, in our expectations. Kumar, et al., 2008 identified the key underlying motivations for the IPRE project:

1. Let the needs of the curriculum drive the design of the robot
2. Use tools that are easy to use, scale with experience
3. Treat the robot as a peripheral
4. Create an accessible, engaging environment for a new, diverse population of students
5. Computer Science is not just Programming
6. Make Computing a Social Activity
7. Make computing a medium for creativity
8. Performances vs. competitions

One of the driving forces underlying the redesign of the introductory course was to address the enrollment crisis in computing. It was therefore obvious to us that in order to attract students from a broader range of interests and disciplines we had to address the perception of the old-style introductory course. Traditionally, the introductory computing courses have been designed to attract students who desire to major in computer science and will ultimately go on into careers in some aspect of software design and engineering. Students who did not meet or test up to that criteria were "weeded out" of the discipline (Patterson, 2006). In the IPRE project, we identified the above motivations as a basis for the redesign so as to provide a much broader view of computing.

During the development of the IPRE project we have discovered that many of the motivations that drive our project have turned out to have unanticipated side-effects related to teaching, course organization, and assessment. In this paper, we focus on the issues related to embracing computing as a medium for creativity.

Creativity and Chaos in Computing

Treating computing as a medium for creativity enables introductory students to write programs and algorithms in service of visual and aural aesthetics. Students write programs to perform choreographed dances for the robots, use the robots to create drawings and sketches, create programs to perform robot plays, use them as electronic puppets, write programs to draw graphics on a computer screen, write programs to compose and play music (Misra, et al., 2009), develop simple games (Xu, et al., 2007), etc. Some of these ideas have been adopted from the media computation approach (Guzdial, 2003). To reduce the learning curve typically involved in using devices like robots, joysticks, and even drawing graphics, we took great care in designing a pedagogically scalable API (Blank, 2006) that enabled a conceptually sound basis for creating and manipulating different kinds of devices and media.

However, one of the things we did not quite account for was the methodology or the process of doing creative work. To us, we were still trying to teach programming as a way of doing computing. However, independent of the methodology of designing and creating programs, we (re-)discovered that in order to do creative work, one needs to provide an environment that is more akin to an artist's studio as opposed to a computer lab. An artist in a studio tends to dabble in several projects or ideas at the same time. There are unfinished pieces of work strewn all over the place. Many of these pieces never reach completion. Of the many completed pieces, only a very small percentage are even considered by the artist themselves to be worthy of "releasing" to the public. More importantly, during the process of creating a finished piece of work, the techniques and the process underlying it, is not necessarily well structured and at times not too disciplined. The finished piece of work is hardly ever an output of a well-engineered, well-planned, process. Invariably, there are blemishes, fixes, paint-overs, on the fly redesigns, etc. That is, the process of creating a piece of art involves several trials, errors, fixes, and redesigns. Most of the time, these become an integral part of the finished work. In other words, there is plenty of chaos in the creative process.

Most prevailing methodologies for teaching introductory programming explicitly attempt to prevent any chaos from the programming process. At the same time, it is interesting to note that there are several parallels here with the programming process. Creating programs involves several trials, errors, bug fixes, and redesigns. In the teaching of computing it is essential to inculcate the idea of trying out stuff ("What happens if I do this?"). However, much of the principles underlying introductory computing pedagogy have moved to intrinsically enforce the use of well structured design principles. Most of these are a trickle-down from software design and engineering principles. We feel that enforcing strict design discipline in an introductory computing course is in direct conflict with some of the motivations we outlined above. In other words, if we want to be successful at creating the perception that computing is a medium for creativity, we have to embrace the chaos that is inherent with doing creative work.

Programming and Chaos?

Enabling creativity as an outcome of a computing activity has direct ramifications to the process of programming itself. This can be seen in the manner that students wrestle with a programming task. At one end of the spectrum one can give completely open-ended tasks. One such task that we have typically assigned in the first few weeks is the goal to make one's robot dance. However, this is not their first assignment. The first assignments show the students how to write a simple function without parameters, and perform a series of discrete actions, such as:

```
def main():
    # turn left at full power for .3 seconds
    turnLeft(1, .3)
    # go forward at full power for 2 seconds
    forward(1, 2)
```

Creating a dancing robot is no more (and no less) than putting together a sequence of calling these functions. As this is extremely early in the course, the students have not seen any type of loop structure. However, students can bring to bear what they know about textual processing and can discover on their own a method of looping: cut-and-paste. In this manner, they can create a robot that can "square dance" by copy-and-pasting a call to the main function 4 times. In addition, many students appreciate that "dancing" is not just any sequence of random movements, and begin to inquire about techniques to group and repeat movements.

In this respect we have come to realize that in such activity we should give the students the freedom to do what comes natural. This applies especially to the way they write more structured programs. As an example, consider the following pseudocode that defines their next assignment:

1. Have your robot perform a dance
2. Ask the user if they would like to see another dance
3. If yes, then repeat

This task builds on their completely open-ended task of the dancing robot, but embeds it in some structure. Most students, of course, would work their way down through the pseudocode sequentially and, quite naturally, end up with something similar to the following in the Python programming language:

```
def main():
   robotDance()
   response = ask("Want to see the robot
                dance again?", ["Yes", "No"])
   if response == "Yes":
      main()
```

Arguably, this is not a "good style" for technical reasons: Python, like many C-based languages doesn't have a notion of proper tail-call position, and this will overflow the program call stack if run for more than 1,000 times. However, it more closely matches the pseudocode and their own intuitions. In fact, it embraces the abstractions of computer science. We find that simple recursion isn't hard for these novice students, but is rather quite natural. Many students discover it on their own.

Unfortunately, the examples that we originally provided the students did not match their intuitions, nor the pseudocode. For example, consider this example using an alternative while-loop and exit flag:

```
done = False
while not done:
   robotDance()
   response = ask("Want to see the robot
                dance again?", ["Yes", "No"])
   if response == "No":
      done = True
```

Our "more proper" example is not as natural as the recursive counterpart, nor does it match the pseudocode. As a result, we have begun to teach the natural use of recursion as the preferred method as it makes more sense to them.

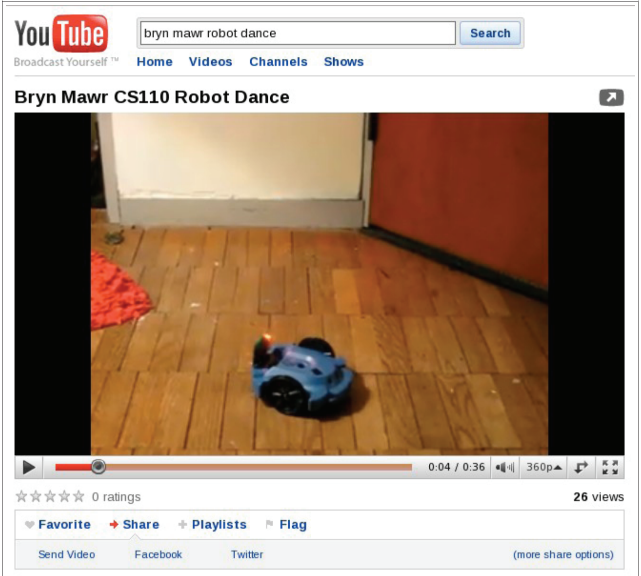

Figure 2: A student's robot dance assignment on YouTube.com.

Therefore, we fully support the use of copy-and-paste, possibly dangerous uses of recursion, and generally whatever spaghetti code that they can cook up. Have we eschewed elegance in programming in service of the creative process? Perhaps. When the end result is a program's creative output is it important that the underlying program be elegant and certifiably well-structured? Visualize a masterpiece painting in a painter's studio soon after putting the final stroke. Would the painting be evaluated for its aesthetics in the context of the entire painting process and the chaos of the studio? Or would it be simply admired as is when hung on a wall in a museum? There are obviously very different criteria at work here. Many might argue that the successful artists have a structure and a discipline to their creative output. We agree. However, we will argue that the so called structure and discipline is different for every artist. It is a very personal thing. Those who copy another artist's structure are not necessarily as successful. This is contradictory to the process of programming in software engineering. By inviting more students into computing and making it a creative process we are trying to make computing a personal activity. And that invited every student's personal chaos into the process.

The role of Fun in Educational Robotics

Our overall methodology could perhaps be put into the category of approaches that value "fun" over substance. Fun approaches are often criticized on a number of points. For example, Fisher (2008) considers that students could experience the feeling of "bait-and-switch" if the introductory courses are based on "fun" but upper-level courses are completely different.

However, even if "fun" is accepted as a valid topic to explore in the classroom, how can one assess it? This is a question with which we all are apparently wrestling, sometimes evaluating it very differently. Recently, a reviewer of one of our papers and video submission had this to say: "The attached video shows only students having fun (which is nice) but does not provide any relevant information." Although there has been some research on the effects of learning and fun in computer science (e.g., (Curzon, 2007) and (MacFarlane, Sim, and Horton, 2005)) we believe that this has yet to be taken seriously as relevant information in education. Learning and play has a long history in learning (including Plato and Freud).

There are some well-known projects in computing that embrace fun out-right. For example, Paul Curzon and Peter McOwan's Computer Science For Fun. They have developed, and edit, the website cs4fn.org and the associated magazine (Curzon, 2007). But these programs are rare.

We must also note that even though we may promote creativity over some perceived notion of proper programming style, this does not necessarily come at the expense of computer science knowledge. For example, Georgia Tech has found no significant difference between sections of an introductory computing course taught with robots and those without (Kumar, et al. 2008).

We believe that "fun" is really a side-effect: students having fun is caused by focusing on their creativity rather than on other aspects of computing or robotics. By fully embracing creativity as the driving motivation, we must relax our constraints on what we think we are teaching.

Assessing: Welcome to the Jungle

There is much emphasis placed these days on assessment of pedagogical initiatives. Without a concrete assessment plan one is unlikely to be successful in obtaining financial support for funding for new curricular initiatives. In our IPRE project, we have performed before and after evaluations for all offerings of our introductory courses. We have shown that using our approach over ¾ of the students are in full agreement about our assessment goals: that they were motivated; they learned computing; and can see that there is an integral role for computing in their future studies. In the past three years, at Bryn Mawr College, we have gone from enrolling less than 5% of our student body (of ~ 1300 students) in the introductory course each year to enrolling over 10% in the same course at the end of three years. Enrollment in our CS2 course and other upper-level computer science courses has more than doubled. We would be hard pressed to find 140 women studying introductory computing each year at even the largest universities in this country. Most funding agencies (and the education community) would be more than satisfied with such an outcome from assessment.

Conclusions and Future Work

We have always found that open-ended robot assignments, such as making a robot dance, have been engaging assignments for the students (if not perplexing to grade for the instructor). However, we have just recently found that taking a relaxed perspective on what kind of code students are writing in the introductory course has allowed the students to focus completely on the creative act. Without proscribing proper ways that they should be writing their code, we often times find that they solve their self-driven problems in a creative way (such as cutting and pasting) or re-inventing great ideas (such as recursion). Furthermore, students are often motivated about wanting to know more about computing as they discover (on their own) the problems of their own design. For example, attempting to debug their own spaghetti-code can be frustrating for them. Allowing students to "babble" in code may be a necessary allowance so that they can still be creative at this early stage. Otherwise, we may end up short-circuiting the pedagogical process, and may lose potential computer scientists.

No doubt, such chaos is not for everyone, student and instructor alike (Dalke, et al., 2007). In fact, we have begun to ask meta-assessment questions such as: how does the instructor's comfort level play a role in the success in a course? We have just begun longitudinal studies of the impact of the instructor's views in the robotics curriculum. However, there is much work to be done in exploring, and exploiting, a student's creativity with educational robotics.

Acknowledgments

This work was funded in part by NSF-0920539, Personal Robots for CS1: Next Steps for an Engaging Pedagogical Framework.

References

Blank, D. 2006. Robots make computer science personal. In *Communications of the ACM*, vol. 49, pp. 25-27.

Blank, D. and Kumar, D. 2002. Patterns of curriculum design. In *Informatics curricula and teaching methods: IFIP TC3/WG3.2 Conference on Informatics Curricula, Teaching Methods, and Best Practice* (ICTEM 2002), July

10-12, 2002, Florianópolis, SC, Brazil.

Curzon, P. 2007. Serious fun in computer science. In *ITiCSE '07: Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*. ISBN 978-1-59593-610-3. Dundee, Scotland. ACM, New York, NY, USA.

Dalke, A.F. Cassidy, K., Grobstein, P. and Blank, D. 2007. Emergent pedagogy: learning to enjoy the uncontrollable—and make it productive. In *Journal of Educational Change*, Volume 8, Number 2 / June, 2007.

Fisher, D. H. 2008. Can AI help develop socially-engaged computational thinkers? In *AAAI Spring Symposium Using AI to Motivate Greater Participation in Computer Science*.

Guzdial, M. 2003. A Media Computation Course for Non-Majors. In *Proceedings of the ITiCSE 2003 Conference,* Thessaloniki, Greece. ACM Press, 2003.

Kumar, D., Blank, D., Balch, T., O'Hara, K., Guzdial M., and Tansley, S. 2008. Engaging Computing Students with AI and Robotics. In *AAAI Spring Symposium, Using AI to Motivate Greater Participation in Computer Science*.

MacFarlane, S. and Sim, G. and Horton, M. 2005. Assessing usability and fun in educational software. In *IDC '05: Proceedings of the 2005 conference on Interaction design and children*. ISBN 1-59593-096-5, pages 103-109. Boulder, Colorado. ACM, New York, NY, USA.

Martin, F. 2007. Real Robots Don't Drive Straight. In *AAAI Spring Symposium, Robots and Robot Venues: Resources for AI Education*. American Association for Artificial Intelligence.

Misra, A. Blank, D. and Kumar, D. 2009. A Music Context for Teaching Introductory Computing. In ITiCSE '09: Proceedings of the 14th annual ACM SIGCSE conference on Innovation and technology in computer science education. pp. 248—252. Paris, France. ACM, New York, NY, USA.

Patterson, D. A. 2006. Computer Science Education in the 21st Century. In *Communications of the ACM*, vol. 49(3), pp 27-30.

Xu, D., Blank, D., Kumar, D. (2008) Games, Robots, and Robot Games: Complementary Contexts for Introductory Computing Education. In *GDCSE '08: Proceedings of the 3rd international conference on Game development in computer science education*. Miami, Florida, ACM, New York, NY, USA.