Concrete Teaching: Hooks and Props as Instructional Technology

Owen Astrachan
Duke University
Department of Computer Science
Durham, NC, USA
ola@cs.duke.edu

1. ABSTRACT

Hooks and props are mental or physical images used in the classroom which help students as they learn new topics. Concrete and constructive teaching are essential in introductory programming courses. In this paper we discuss active teaching and some ideas using physical props and images that in our experience have enhanced the teaching and learning process.

1.1 Keywords

Active learning, active teaching, introductory computer science, C++, Java, pointers.

2. INTRODUCTION

There has been significant recent work on using technology to enhance the educational process. Much of this work focuses on developing online courseware, e.g., [5, 6], labs, e.g., [4, 2], and subject-specific support using, for example, visualization [1, 11]. Research into different styles of learning has focused recently on learner-centered design [3, 8, 13] which emphasizes (among other things) a constructionist theory of learning wherein students learn through a process of building their own mental models. Facilitating mental model-building with well-crafted instructional software, labs, and curricula is an important component in improving education.

Technology and the web are perhaps the most important tools with which we can augment and change what and how students learn. In this paper we argue that providing concrete, constructive hooks, using props were possible, is necessary for students to get the most out of the teaching process. Here we contrast the teaching process, directed by a guide or teacher, with the learning process, where we agree that students are their own best guide. A hook is a mental or physical image on which students can hang their understanding as they learn a new topic. Concrete and constructive teaching is crucial in introductory programming courses where students must build mental models of language constructs to become proficient or accomplished programmers and designers. Visualization and courseware are crucial in reaching this goal, but teacher-directed instruction is an important component of successful introductory programming courses. The work reported in [7] emphasizes active learning in the classroom. We espouse this approach as well, in fact it is crucial. In this paper we discuss methods that help teachers augment active learning with hooks implemented using bare-bones technology to enhance the learning and teaching process.

Physics classes, for example, often use demonstrations to provide a hook for student understanding. For example, the classic monkey gun experiment illustrates how gravity affects falling objects. In [7], McConnell uses the term *physical activity*, and illustrates objects and classes using paper bags containing private data. This hook, in the shape of a physical and mental image students use both consciously and unconsciously, is an example of the methods we report on here. We provide three examples of these physical hooks and an example of a metaphorical approach to explaining programming concepts. Metaphors are also important in didactic learning. In [9], for example, an explanation of a well-engineered program in terms of stereo components shows how functions fit together using parameters as stereo components fit together through well-defined component interfaces and wires.

3. Parameter Passing

In our introductory courses we use C++. For the purpose of this section, the only important feature of the language is that there are three modes for passing parameters: by-value, by-reference, and by-const-reference. In contrast, Pascal has two modes of passing parameters: by-value and by-reference, C and Java have only one mode: by value[1],

[1] Note that all Java parameters are references/pointers, but the parameters are passed by value, i.e., it is not possible to modify a parameter by assignment and have the assignment have an effect outside of the function in which the assignment takes

and Ada has (at least) three: in, out, in-out. Our students have little difficulty understanding parameter passing when we use only pass-by-value, but often have problems understanding when we use the other modes in C++, and when we discuss aliasing. Interestingly, we did not have the same problems with parameters when we used C as when we used Pascal or C++.

In our introductory course we have a lab specifically designed for students to discover the differences among the three parameter passing modes on their own. We use group exercises in class were students analyze and predict the behavior of functions written with different parameter passing modes. However, one active presentation gives students a physical experience as well as a mental model that augments the other learning activities. We represent variables and parameters as Frisbees (flying discs). We bring several sizes and colors of Frisbee to class when we perform this in-class demonstration so that we can represent different types, e.g., string, int, object. In the active demo, one student represents a function and another student represents a code block that calls the function. We begin by explaining that storage for value parameters is owned by the function. To call the function, the caller gets the parameter/Frisbee from the function, copies a value onto the Frisbee by either writing on the Frisbee or using a post-it, and passes the parameter. Of course the exchange of parameters is realized by flying the parameters between the caller and the function. In the demo we use different functions, not all of which modify the parameter. Since the parameter is a local copy owned by the function, the modifications have no affect in the calling code.

When passing parameters by reference, storage is owned by the calling code. To pass a parameter, the function gets the storage and value for the parameter from the calling code. Usually we pass a piece of string attached to the parameter so that the parameter itself stays with the caller. This allows us to attach more than one piece of string to the same parameter to illustrate the affects of aliasing in parameter passing. Usually we pass the parameter first and then use the string to model the actual process more closely. The piece of string is also the same size regardless of the size of the parameter. This reinforces pass-by-reference for efficiency concerns. C++ has the nice feature of allowing pass-by-reference for efficiency to be combined with const for safety (non modifiability by the called function). To illustrate this mode we enclose the Frisbee in a plastic bag and throw the bag (this is before we use the string-and-Frisbee model of reference parameters). The called function examines the Frisbee, but the plastic bag is protection against modifying the parameter.

4. Linked Lists

A significant portion of our second course, and part of our first course, is spent on helping students master linked data structures, specifically lists and trees. We use traditional "box-and-pointer" diagrams, augmented with programs and labs to illustrate how different operations affect linked lists, e.g., adding a node, deleting a node, traversing a list. We also use ddd, a debugger that provides a graphical representation of linked structures which students can resize, move, and inspect. The use of the debugger is essential in providing students with a visual model of what their code is doing. We have used visualizations [10, 14] to help as well, but use a hook in a physical demo that is a proven success in providing students with the same kind of mental and physical model that Frisbees provide for parameter passing.

We use plastic building blocks designed for small children. These blocks come in different colors, with different shapes, and have beaded ends that allow the blocks to be plugged into each other. Figure 1 shows these props in action, one as a circularly linked-list and one as a plain linked list.

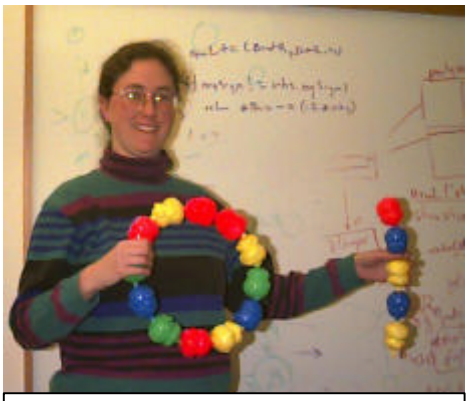

Figure 1. Building Block as Linked

Certainly these props are not essential to student understanding of linked structures, but they provide a physical hook on which students can build a mental model of how linked structures work. We think the physical nature of the blocks is an important addition to visualizations and simulations that we also use.

5. Pointers

Students often have a difficult time with pointers in C++ and references in Java[2]. To illustrate what happens when *new* is called in either C++ or Java, we use a physical representation of a pointer made out of a sticky, stretchable rubber-like substance commercially marketed as "icky-poo". We use different colors of icky-poo to represent

place. In addition to enlivening the classroom discussion of parameters, this demonstration provides a physical and mental hook that students use in subsequent labs and programming exercises.

[2] In Java references are essentially pointers, but since there is no address-of operator in Java and since garbage collection makes it unnecessary to delete/free memory, students do not encounter some of the same problems as they do in C++.

typed pointers and references in C++ and Java, respectively. In code, a pointer definition without a value assignment results in a dangling pointer convincingly shown as a wiggly, shaking pointer. A null pointer is shown hanging downward, all pointers can point to this "special place" which facilitates pointer comparison without memory allocation. Finally, calling new returns a chunk of memory that is literally attached to a pointer. In our class demo we use different colored or sized paper to represent different types of memory, e.g., int and string.

Calling new causes the teacher to cast the pointer (no pun intended, here cast is used in the sense of a fishing line) towards the pool of memory. A good cast results in the larger end of the pointer becoming attached to a piece of paper, which is then snapped back to the caster because of the elastic properties of the pointer. A bad cast yields no memory engendering a discussion of whether this results in a null pointer or an exception. In Figure 2 both null (hanging straight down) and successfully allocated (attached to the board) pointers are shown. Again the physical nature of this demo, combined with the actual demonstration which typically results in a few misses, but well-cheered successful allocations, is an important facet of other methods we use for students to learn about pointers.

Figure 2. null, non-null pointers

6. Outside Readings as Metaphors

We use metaphors in addition to physical props. Often these metaphors are found in places one might not expect. For example, *The Cat in the Hat Comes Back* [12] has a fine example of recursion including a base-case (a small cat named Voom) used to clean up red snow. In our classes we read this book aloud on the last day of class. This is always a resounding success bringing a different kind of understanding to students. We have several kinds of students in our introductory courses: those who continue with programming and computer science and those for whom the course is their last exposure to the field; all these students remember the end-of-class reading. We do not advocate sacrificing rigor and understanding for show. However, when a show can reinforce the rigor, we believe it is important for students to remember that learning and class are enjoyable and that topics from computer science can be found everywhere.

Other Dr. Seuss books are used in our operating systems course to explain deadlock (*The Lorax*), and in our introductory courses to explain linked lists (*I can Lick Thirty Tigers Today*).

7. Summary

Using technology in the classroom, and as part of the teaching and learning process, is essential in the current environment. Too often, however, we forget that explanation-based teaching does have a place in the classroom, and that some lecturing can be a valuable addition to group exercises, labs, and interactive learning. Props and metaphors should be used to augment the classroom experience and to provide a physical image that helps students construct a mental model.

8. REFERENCES

[1] ASTRACHAN, O., AND RODGER, S. Animation, visualization, and interaction in CS 1 assignments. In *The Papers of the Twenty-ninth SIGCSE Technical Symposium on Computer Science Education* (1998).

[2] BALDWIN, D. Three years experience with gateway labs. *In Proceedings of the 1996 ITCSE conference* (1996), ACM/SIGCSE, pp. 6-7.

[3] JACKSON, S. L., KRAJCIK, J., AND SOLOWAY, E. The design of guided learner-adaptable scaffolding in interactive learning environments. In *Proceedings of CHI 98* (1998).

[4] KNOX, D., WOLZ, U. *et al*. Use of laboratories in Computer Science education: guidelines for good practice. In *Proceedings of the 1996 ITCSE conference* (1996), ACM/SIGCSE, pp. 167-181.

[5] LAWHEAD, P. B. A model for the creation of online courseware. *In Proceedings of ITCSE 97* (1997), ACM/SIGCSE, pp. 31-36. SIGCSE Bulletin, September 1997.

[6] MARSHALL, A., AND HURLEY, S. Interactive hypermedia courseware for the world wide web. In *Proceedings of the 1996 ITCSE conference* (1996), ACM/SIGCSE, pp. 1-5.

[7] MCCONNELL, J. J. Active learning and its use in Computer Science. In *Proceedings of the 1996 ITCSE conference* (1996), ACM/SIGCSE, pp. 51-54.

[8] PAPERT, S. *The Children's Machine (Rethinking School in the Age of the Computer).* Basic Books, 1993.

[9] REGES, S. *Building Pascal Programs*. Little, Brown and Company, 1987.

[10] RODGER, S. An interactive lecture approach to teaching computer science. *In The Papers of the Twenty-Sixth SIGCSE Technical Symposium on Computer Science Education* (March 1995), ACM Press.

[11] RODGER, S. H. Integrating animations into courses. *In ACM SIGCSE/SIGCUE Conference on Integrating Technology in Computer Science Education* (Barcelona) (1996), pp. 72-74.

[12] SEUSS, D. *The Cat in the Hat Comes Back*. Beginner Books, 1986.

[13] SOLOWAY, E., JACKSON, S.L., KLEIN, J., QUINTANA, C., REED, J., SPITULNIK, J., STRATFORD, S.J., AND STUDER, S. Learning theory in practice: Case studies of learner centered design. In http://hi-ce.eecs.umich.edu/papers (1997).

[14] STASKO, J. Using student-built algorithm animations as learning aids. *In The Papers of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education* (1997), pp. 25-29.