Constructivist Implications of Preconceptions in Computing

Kris D. Powers[1]
Computer Science Department, University of Illinois at Springfield
Springfield, IL 62794-9243, U.S.A.

and

Daniel T. Powers[2]
836 N. Cooper St.
Peoria, IL 61606, U.S.A.

Abstract

The theory of constructivism has several important implications for methods of teaching. One of these is the need to explicitly confront student preconceptions. In this paper we explain how preconceptions effect student learning, according to the constructivist view, present an initial collection of preconceptions which computer science educators must address, and discuss how identifying these preconceptions can help improve student learning in CSIS.

Keywords: Teaching methods, constructivism, preconceptions.

1. INTRODUCTION

The theory of constructivism has several important implications for methods of teaching. One of these is the need to explicitly confront student preconceptions. In other disciplines, extensive research has been devoted to uncovering the student preconceptions which educators must address. Indeed, numerous volumes have been dedicated to expositions on student preconceptions within particular disciplines, such as physics and mathematics. We believe that uncovering student preconceptions about computing is a vital step for improving Computer Science and Information Systems (CSIS) education along constructivist lines. As has been demonstrated in other disciplines, once student ideas are identified and understood, they can be used to improve student learning by helping to guide what concepts are taught and at what point in the curriculum, and how the learning experiences for particular concepts may be adapted. (Driver 1985) In this paper we explain how preconceptions effect student learning, according to the constructivist view, present an initial collection of preconceptions which CSIS educators must address, and discuss how identifying these preconceptions can help improve student learning.

Context
Little work has focused on identifying the initial ideas that students bring with them to the door of their first computing class. Some related work on student conceptions has been done, but it differs from ours in the following important ways. First, these works primarily consider the conceptions that students construct once in the CSIS classroom, not the conceptions that they bring with them to the door. Also, most of this work is limited in scope to programming per se, as opposed to CSIS generally. (See, e.g., Chang 1994.)

The research most closely related to this would be the work of Ben-Ari, which appeared when this work was in its infancy (Ben-Ari 1998). Ben-Ari surveyed the theory of constructivism generally, and showed how it could be used to analyze particular issues in CSIS education (e.g.,

[1] powers.kris@uis.edu
[2] dtp@hotbot.com

"GUI and WYSIWIG Angst"), and naturally included some discussion of student conceptions. Our work has a more narrow focus. The need to address preconceptions is only one of several implications for methods of teaching that result from the theory of constructivism. We discuss a set of preconceptions which are fundamental to CSIS education and consider their impact on classroom teaching and curriculum development. While the difficulties presented by many of these preconceptions are not new to seasoned CSIS educators, the techniques and implications for addressing them along constructivist lines often are.

Finally, the discussion of student preconceptions is an important *recurring* issue in CSIS education. In other disciplines, such discussions are not subject to the rapid pace of technology. For example, student preconceptions of Newtonian mechanics have probably not changed drastically in the last twenty-five years. In contrast, most of us would probably agree that student conceptions of computing have varied greatly over the same time period. Much existing work on student conceptions was performed a decade or more ago, and our hope is to help reinvigorate this strand of research.

The Theory Of Constructivism

For more than a decade, educators and psychologists have been using the theory of constructivism to explain learning. Do they know the real truth? No – because according to constructivism, no one can. Constructivism holds that while there is a physical reality, we can never say that what we know is the truth because all of our knowledge has been constructed from our own personal experiences and social interactions in a particular cultural setting. Since no one's experience is complete, no one's knowledge is complete.

"Free knowledge – bring your own container." This little maxim is more than just a classic classroom poster; it's the way we like to think that we teach. We dispense wisdom, and the wise will soak it up. The students are empty receptacles, or, if not, what we tell them is so shiny and new that it will undoubtedly replace all of those childish notions that they brought with them. But constructivists tell us that it is just not so. The old knowledge affects the new. Any new knowledge our students construct in response to new experience will be incorporated into the framework of knowledge they have already constructed.

The theory of constructivism has several important implications for methods of teaching. The knowledge the students have already constructed is based upon previous experiences. If the learning we attempt to provide has no basis in experience, it has little chance of modifying that which the students already "know." So one implication is that our teaching must be *experiential* to be effective. Also, because their perceptions explain what they have experienced, students believe that this knowledge is correct. Constructivists agree. Even if the ideas seem ludicrous, they are merely naive – based on incomplete experience and a lack of social interactions that would challenge them. Inaccurate ideas "may persist ... despite formal teaching" (Driver 1994, p. 2) It is not enough to teach the correct idea. We as educators must explicitly confront our students' inaccurate *preconceptions* before they can be dispelled. Finally, knowledge is constructed in a *social setting* influenced by the instructor. Within this setting students must be provided with an opportunity to form new knowledge in cooperation and interaction with their peers.

Of these implications for methods of teaching, the importance of experiential, hands-on learning in CSIS is well accepted and is integrated, in some form or another, in many curricula. The principle of experiential learning also provides theoretical support for a number of formal teaching methods. For example, "discovery learning" is a broadly applied term that has been used to describe any activity in which the learners are free to make there own discoveries about a certain phenomenon. Baldwin describes the successful application of discovery learning in two courses in computer science, one graphics and the other a C/UNIX programming course (Baldwin 1996). A more pervasive teaching method termed "problem-based learning" presents students with an ill-structured problem and puts them in the role of problem-solvers while the teacher serves as coach. The most obvious potential for problem-based learning in computing lies in the proliferation of design problems encountered throughout the curriculum.

Of the other implications for methods of teaching, the importance of social interaction is also well recognized within the CSIS discipline. In particular, working in teams is a part of educational experiences advocated by *Computing Curricula 91*, and our students' inability to work in teams has been one of the main criticisms against computing education. However, this recognition is based on the importance of the activity as an *end*, not as a *means* to an end. According to the constructivist approach, students must assimilate new scientific knowledge into their existing frameworks in order to effectively form and express their own opinions, and engage their classmates in discussion. The social interaction is the catalyst for acquiring new knowledge; it is not the knowledge itself.

The constructivist view that new knowledge is formed through social interaction is the basis for another major, well-studied teaching method: "cooperative learning." Sabin and Sabin described a successful application of cooperative learning for teaching introductory programming. (Sabin 1994) In this application students worked cooperatively in class to solve smaller problems, typically related to newly introduced material. The work of Daigle, et. al. (Daigle 1996) described exercises for collaborative learning throughout the curriculum. The constructivist view has also been used for other less pervasive teaching methods, like the Science-

Technology-Society theme (Bybee 1987). This method holds that students are motivated by the interaction between science, technology and society, and that these areas are necessarily intertwined. Students are asked to consider the ramifications of a particular technology on their lives or on society as a whole. For CSIS, this approach implies that addressing the social and professional contexts of computing, as outlined in *Computing Curricula 1991*, is doubly important. Not only do students engage the social issues important to their field, but in doing so they also crystallize their knowledge of the underlying computing science. For example, a discussion about the social implications of various types of encryption (e.g., strong versus escrowed) can be used as a highly effective springboard for reinforcing the students' knowledge of the related algorithms.

The final implication for methods of teaching which results from the theory of constructivism is the problem of preconceptions. We focus our attention on it in the next section.

2. PRECONCEPTIONS OF COMPUTING

Considerable research has focused on the erroneous ideas that beginners develop in the process of learning to program. But to our knowledge, the idea of considering the intellectual framework that existed *before* the learner engaged the subject has not been explored. To what extent might their erroneous ideas be the result of general knowledge, formed in the general social setting, which is either inaccurate or misapplied? CSIS educators must confront the erroneous preconceptions that students bring to the discipline from general society, and these preconceptions cannot be confronted until they are identified. In this section, we present a collection of fundamental preconceptions in CSIS as a first step in this task. We do not pretend to identify new hurdles which CSIS educators must face, but rather propose how certain ones may be viewed through the constructivist prism.

CSIS == coding

The difficulty of preconceptions is all too readily demonstrated by many students' preconception of what the computing discipline is: coding. Pop culture endorses an image of an archtypical computer "geek" that is all too readily accepted: a solitary hacker too obsessed with writing code to be bothered with social interaction. But even the most discerning individuals, who reject the stereotypical dress and behavior of this image, still accept the notion that computing professionals do nothing but write code. The high visibility of current recruitment efforts for programmers has reinforced this notion. We are all well aware of the efforts that have been made to dispel this myth, but as is classic in the theory of constructivism, inaccurate ideas are not easily dispelled.

Computers as analogue devices

As was noted by Dijkstra, another problem encountered by those engaging computing for the first time is expecting the behavior of computers to mirror that of familiar analogue devices (Dijkstra 1989). Students are accustomed to devices that respond linearly with variances in their input. For instance, the accelerator or brake in a car changes the speed of the car in response to the degree of pressure on the pedal. Based on this type of experience, a novice programming student would expect that a program that is "nearly" correct will produce output that is likewise "nearly" correct. The fact that changing a single line of code, indeed even just changing a single bit in certain cases, can drastically alter the output runs counter to their analogue experiences. Demonstrating the reality of such a possibility can helps students deal with debugging down the road.

Computing through trial and error

Since the presence of computers has become ubiquitous in our society, the young have been more successful and comfortable using computers precisely because they eagerly engage in trial and error. "How do I make a table in MS Word? Oooo.. Let's try THAT button!" Adults are more likely to sit and read the user manual before trying. The idea that finding the most successful ways to use a computer occurs through trial and error seems pervasive, and may well be correct. However, students are then apt to try the same approach with every activity involving the computer, in particular coding. Computer use as a trial and error activity leads to coding as a trial and error activity - i.e., hacking.

Technological details are primary knowledge

Those regarded as most technologically literate in everyday culture seem to be those who are most facile with the details of currently available products. As a result, students tend to attribute an artificial importance to the ephemeral details of technology dependent information. The knowledge of the processes and concepts of the field that transcend those details may become obscured. Driver, et. al. point out that one implication of preconceptions is that "pupils may reinterpret the intentions of the teacher in terms of their own understanding." (Driver 1985, p. 7) For example, an instructor's use of a sample architecture to teach a computer organization class may be misinterpreted by her students. Unless the purpose is clear, a student may well miss the big picture, and end up focused on the details of the particular system.

Algebra class math == computer math

> "When is X+1 not 1 bigger than X? Never, of course!"
> "X = X + 1? Don't be silly!"

Obviously our students bring with them years of mathematics education in which they have built complex models and frameworks for their understanding. Many of these frameworks have a high degree of functional accuracy. But it is precisely this accuracy that can stymie learning efforts. For example, after weeks studying data representations and two's complement numbers, a student was baffled by a negative number in his program. "My variable starts out with a positive value, and I only ever add 1 to it! How could its value become negative?" Also, the meaning of many seemingly mathematical expressions (e.g., assignment statements) can be different in the context of computing.

Complexity and levels of abstraction
Teaching students to deal with the complexity of computing systems through top-down design and step-wise refinement are standard fare in introductory programming courses. But few students ever truly grasp the degree of complexity of computing systems and the absolute necessity for levels of abstraction throughout computing. Indeed, students often equate the complexity of computers with other much simpler digital devices, such as telephones and stereos. Dijkstra asserts that such student conceptions of the complexity of computers are "orders of magnitude worse than comparing ... the supersonic jet plane with a crawling baby..." (Dijkstra 1990, p. 1400) When confronted with the differential in complexity, a computer architecture students remarked "I used to get [angry] when my computer crashed. But, the more I find out about what's going on inside, the more amazed I am that it doesn't just crash all the time." Failure to confront our students' conceptions may well be what prevents them from embracing the techniques of abstraction we are trying to impart.

Computer concept
Because computers are increasingly commonplace, most students have some concept of what a computer is. However, these concepts range from those that are very accurate to those more akin to a box of monkeys (as seen in cartoons) or a giant brain. The simplest of these preconceptions, like the giant brain, oftentimes seem to be a direct reflection of the student's perception of the capabilities of the machine. For instance, the sophistication of today's user interfaces inspires a myriad of preconceptions regarding the capabilities of the machine. Consider screen images which concurrently present information from multiple applications. The reality that this information is distinct and not necessarily shared within the computer is not apparent to students. The result may be a "giant brain" conceptualization of the machine, in which all parts are "aware" of all the information available. "Why do I need to write code in my program to determine the date? The computer already knows that!"

As the range of possible preconceptions is considered, a critical factor becomes the extent to which student concepts may be considered "effective" or "viable" for explaining the totality of their (limited) experiences. For example, Harvard University's Project Star found that most students, alumni and faculty members questioned about seasonal variations in temperature asserted the belief that they were caused by the Earth's position in its elliptical orbit (Schneps 1987). These same individuals were probably quite well aware that the temperature near the equator remains almost constant year-round. Thus, their model was not "effective," as it did not incorporate all of their pertinent knowledge. Ben-Ari contends that most of our students, in fact, lack an effective concept of what a computer is. "If misconceptions are essential to the construction of new knowledge, the lack of an effective, if flawed, model of a computer is a serious obstacle to learning CS." (Ben-Ari 1998, p. 259) The current absence of an "effective model" is certainly a debatable and dynamic point. However, it seems clear that this situation is changing. Ben-Ari agrees: "As computer literacy becomes common, if not universal, students will begin their academic studies with an effective model of a computer." (Ben-Ari 1998, p. 261) Identifying when we have reached this juncture, and what model is commonly held is critical to computing education.

However, the impact that a student's "computer concept" has on their learning is not limited to whether or not it is effective; non-effective preconceptions can significantly hamper student learning as well. While the models for seasonal variations in temperature held by the participants in the previously mentioned Project Star (Schneps 1987) were ineffective, this does not mean that these models did not have to be addressed in teaching. Dijkstra asserts the situation in CSIS is particularly troublesome. He begins by stating that "one has to approach a radical novelty with a blank mind, consciously refusing to try to link history with what is already familiar..." (Dijkstra 1989, p. 1398), and goes on to contend that "computers represent a radical novelty, and that only by identifying them as such can we identify all the nonsense, the misconception, and mythology that surround them" (Dijkstra 1989, p. 1399). In other words, students must first be forced to abandon the totality of their preconceptions in order to learn CSIS.

3. IMPROVING STUDENT LEARNING

In the previous section we presented an initial collection of the preconceptions which may hamper students new to the computer science discipline. How can efforts to identify and understand student preconceptions help improve student learning? Driver, et. al summarize how knowledge of students' preconceptions can help guide the educator in a number of critical pedagogical choices: which concepts to teach and when, what learning experiences to use, and how to present the goals of proposed activities (Driver 1985).

Which concepts to teach and when

"Possible teaching sequences are prepared by analyzing which are the most basic ideas, from a scientific perspective, and building the curriculum from there. ...[such] schemes may make assumptions that [our students] have already constructed certain basic ideas and this may well not be the case. ... in curriculum planning it is necessary not only to consider the structure of the subject but also to take into account the learner's ideas. This may mean revising what we consider to be the starting points in our teaching – the ideas we can assume pupils have available to them." (Driver 1985, p. 199) For example, a constructivist solution to the student perception that *CSIS == coding* is to adopt a curriculum beginning with a breadth-first introduction which defines the field. The instructor must actively and explicitly teach that *CSIS != Coding*. That is, the instructor must not merely teach what CSIS *is*, but also teach what it is *not*. In another example, Ben-Ari claims that students enter the classroom with no effective model of a computer, and that this is major hurdle to getting started in CSIS (Ben-Ari 1998). If we accept this hypothesis, then teaching what a computer *is*, before launching into other curricula topics may aid student learning.

What learning experiences to use

Research in other disciplines has shown that there are prevalent preconceptions among students, and that educators can address these common preconceptions through careful selection of appropriate learning experiences. (Tobin 1993) Strategies for accounting for preconceptions in specific learning tasks have been suggested by a number of research studies (Driver 1985):

- *provide students with opportunities to make their ideas explicit*
  Students' mathematical backgrounds make early, fundamental topics like assignments statements difficult to understand. Sequences of code like:
  ```
  x = 1; y = 2; x = 3;
  ```
  have been known to baffle students. When students are asked to verbalize their own accounts of how such statements operate, the confounding preconceptions are made clear. (Chang 1994)
- *introduce discrepant events*
  For instance, hackers not only need laboratories guiding them in structured program development, but also laboratories demonstrating the relative ineffectiveness of ad hoc approaches to problem solving. Or, an obvious way to challenge students regard for computer arithmetic is to have them explain the output of:
  ```
  x = 1;
  while (x > 0) x++;
  cout << "x = " << x << endl;
  ```
- *encourage the generation of a range of conceptual schemes*
  Instead of presenting students with a correct computer concept as a fait accompli, it is more effective for the instructor to develop *with* students a range of possible concepts, effective and not. Improved learning occurs as the instructor and students examine the viability of the concepts together.
- *practice using ideas in a range of situations*
  The importance students tend to attribute to technology dependent information can be demonstrated as secondary to knowledge of the processes and concepts of the field by applying the latter in a variety of contexts. Two areas that are particularly appropriate are computer organization and operating systems. In both areas, students are likely to become too absorbed by the details of a certain example, and fail to generalize the concepts under study.

How to present the goals of proposed activities

This strategy was alluded to previously in the context of preventing technological details from obscuring more important concepts. It can also be applied to presenting the goal of programming exercises. Student fixation on coding tends to equate correct output with satisfactory performance. Addressing this preconception requires explicitly addressing the importance of features other than correctness: style, documentation, etcetera. Exercises that force students to deal with ill-structured and undocumented code may be effective at clarifying the multiplicity of programming goals.

4. FUTURE WORK

As previously stated, CSIS educators must carefully address student preconceptions. Obviously, this cannot be accomplished until these preconceptions are identified. While we have suggested an initial collection, there are certainly many more. Some of these may be readily evidenced by other instructors' experiences. Identifying others will require specialized research.

Of the preconceptions effecting CSIS education, the most important is certainly that of our students' "computer concept." Its precise nature, however, is also one of the most difficult to discern. In order to identify this preconception more accurately, we are currently engaged in a qualitative research study. The first step, already completed, has been a limited interview study of high school seniors, focused on gathering initial insights into student conceptualization of the computer. For instance, one insight garnered from this step supports the notion that the predominant conceptualization of a computer has evolved significantly as its predominant use has changed. In the mid-seventies, computers were equated with numerical calculators and business machines. By the eighties, PC's and word processing changed the view to that of a fancy typewriter. In our current qualitative interview study, the predominant view seems to equate computers with information. One participant of the interview study described a computer as like “having a twenty-four hour, seven day a week library.” From the range of insights provided by these initial interviews, a WWW-based survey will be developed to obtain a more expansive data set. The conjectures that result from this data will be the basis for a final, more thorough interview series.

It should be clearly noted that the end goal is certainly not the mere identification of preconceptions. Once preconceptions are identified, discrepant events and/or other appropriate learning experiences must be developed and disseminated. We hope that others will join in this effort by participating in a WWW repository currently in creation. This repository will disseminate not only those preconceptions we have discussed, but also those contributed by other CSIS instructors. It will likewise serve as a warehouse for specific learning experiences developed to address them, by both the authors and contributing educators.

5. CONCLUSIONS

In this work, we have proposed an initial collection of fundamental preconceptions which CSIS educators must address. We are certainly not proposing that this collection includes the entirety of what may arguably be considered a "fundamental" preconception. For instance, there is a host of preconceptions related to networks and the Internet that could have easily been included in our collection. Our choices were prejudiced toward those ideas that seemed both fundamental and best suited for providing examples in this limited discourse. Because preconceptions tend to be immutable, CSIS educators must explicitly confront erroneous ones in order to prevent students from reverting to them after formal instruction. We have also presented strategies that can be used to address these inaccurate models and others that impede our students' learning. We believe that uncovering student preconceptions about computing is a vital step for improving computing education along constructivist lines.

6. REFERENCES

Baldwin, D., 1996, “Discovery Learning in Computer Science,” Proceedings of SIGCSE '96, March 1996, pp. 222-226.

Ben-Ari, M., 1998, “Constructivism in Computer Science Education,” Proceedings of SIGCSE '98, March 1998, pp. 257-261.

Bybee, R., 1987, “Science Education and the Science-Technlogy-Society (S-T-S) Theme,” Science Education, 71(5), 667-683.

Chang, B., 1994, "A Study on the Analysis of Error Patterns and Misconceptions for BASIC Programming of Novice College Students in Taiwan," Proceedings of the 3rd International Seminar on Misconceptions and Education Strategies in Science and Mathematics, published online at URL: http://www2.ucsc.edu/mlrg/proc3abstract.html.

Computing Curricula 1991: Report of the ACM/IEEE-CS Joint Curriculum Task Force, 1991, ACM Press/IEEE Computer Society Press.

Daigle R., M. Doran and J. Pardue, 1996, “Integrating Collaborative Problem Solving Throughout the Curriculum,” Proceedings of SIGCSE '96, March 1996, pp. 237-241.

Davis, R., C. Maher, and N. Noddings (editors), 1990, “Constructivist Views on the Teaching and Learning of Science,” J. for Research in Mathematics Education, Monograph No. 4, National Council for Teachers of Mathematics, 1990.

Dijkstra, E., 1985, “On the Cruelty of Really Teaching Computer Science,” Communications of the ACM, 32(12), 1398-1404.

Driver, R., E. Guesne, and A. Tiberghien (editors), 1985, Children's Ideas in Science, Open University Press, Philadelphia.

Driver, R., A. Squires, P. Rushworth, and V. Wood-Robinson, 1994, Making Sense of Secondary Science, Routledge, London.

Gabel, D. (editor), 1994, Handbook of Research on Science Teaching and Learning, Simon and Schuster Macmillan, New York.

Pearsall, M. (editor), Volume II: Relevant Research, 1992, National Science Teachers Association, Washington.

Schneps, M., and P. Sadler, 1987, A Private Universe, video, Harvard-Smithsonian Center for Astrophysics Washington, D.C.: Annenberg/CPB Collections.

Sabin, E. and R. Sabin, 1994, “Collaborative Learning in an Introductory Computer Science Course,” Proceedings of SIGCSE '9, March 1994, pp. 304-308.

Steffe, L. and J. Gale, 1995, Constructivism in Education, Lawrence Erlbaum Associates, Hillsdale, NJ.

Tobin, K. (editor), 1993, The Practice of Constructivism in Science Education, Lawrence Erlbaum Associates, Hillsdale, NJ.

Wandersee, J., J. Mintzes, and J. Novak, 1994, "Research on Alternative Conceptions in Science," in (Gabel, 1994), pp. 177-210.