STUDENT CENTERED DESIGN FOR EDUCATIONAL SOFTWARE

by Jere Confrey and Erick Smith
Cornell University

Introduction:

In this paper we wish to suggest that our current ventures into the design of computer software for the exploration of mathematical ideas ought to be viewed as tentative, but necessary first approximations. It is a plea for modesty that we might become aware of how impoverished our own knowledge is about student conceptions, learning, and the act of representing knowledge in a dynamic interactive medium.

In this spirit, we make four claims:

1) Software designed to in some sense give a literal "representation" of mathematical ideas will not be as effective as software designed to assist students in solving problems by providing a medium for them to express and record their mathematical ideas, allowing them the opportunity to reflect on these conceptions.
2) We should expect our first implementation to provide us with opportunities for revision. We need to examine how the students use the software and search out how the deviations from our expectation are evidence of alternative conceptions.
3) If we are open to it, examining these alternative conceptions may lead us to experience transformations in our own understanding of mathematics. These transformations are the result of our awareness of both the student as making viable mathematical statements within the constraints of his/her conceptual world, and of the computer as providing both a different means for mathematical action and for the communication of mathematical ideas.
4) Translating the idea for the program into a specific software program through interaction with our technical designers and programmers as they code it exerts another influence on the product. We can improve design by using tools such as Hypercard™ to storyboard the interfaces and thus reduce the distance between conceptual design and the actual program. This issue will not, however, be further addressed in this paper.

A Conventional View of Software Design:

To contrast our proposal for the design of software from that which seems often to be used, we present a hypothetical model of a typical design process:

1) Select a piece of mathematics to implement.
2) Choose which forms of representation (graphs, tables, equations, matrices, etc.) to use and the appropriate operations for each (numeric calculations, plotting, algebraic manipulation, etc)
3) Consider how to make this mathematics accessible and palatable to the student (user-friendliness) in such variables as speed and frequency of feedback, orientation to the system, attractiveness in the interface, motivational encouragement, etc.
4) Develop design documents for programmers to use to imitate as close as possible the mathematical representations, hiding to whatever extent possible the idiosyncrasies of the computer medium.
5) Complete the product and prepare documentation.

Such a process fails to take into consideration the importance of students' conceptions and the interactions between the student and the computer as an emergent process. Awareness and anticipation of these sources of design implementation can help both in the initial and later design stages. An example is provided to illustrate how these can have a useful impact on design.

Design of the Program ***Function Probe*** ™:

Function Probe™ is a piece of software which allows students to use multiple representations (graphs, tables, equations, and calculator) of functional relationships as tools in their problem-solving endeavors. In implementing the first version of this software, we had two working requirements:

1) Although certain basic functionality was required for each representation, this functionality was designed, modified, expanded, and implemented through an interactive process between educator and programmer in which observed student strategies were used as the development guidelines.
2) Each representation was required to be complete within itself, allowing students to freely follow any desired path in their choice and use of tools in the process of solving problems. In addition, the ability to transfer information among representations was required.

Our claim for modesty in this design is that it does not and cannot have the capability to allow students all the desired freedom and flexibility that would be ideal in problem-solving situations. However, the ultimate judge of how effective we are in approaching those goals and the determination for future modifications lies in the way the students actually use the software, not in how accurately the mathematics is presented or how attractively we create the interface.

Evidence for a Student-Centered Design Approach, an Example:

A single example will be provided of the evidence that students will not use the software as expected, and that these deviations from our expectations form the basis from which good design is made.

The traditional implementation of a logarithmic capability in a software program would be centered around the following curricular goals (precalculus level):

a) translations from exponential to log form
b) rules of logs
c) change of base formula
d) ln x as the inverse of e^x.

and would suggest the implementation of two log functions, log x and ln x. These are the tools and curricular areas which have been developed for working with logs and the most likely ones we would select as the "expert knowledge", "correct representations" and "traditional content" from our domain of mathematics. With the following example, we wish to demonstrate:

1) how a tool can be used to transform our agenda and conception of how such a topic could be taught, and
2) how putting the idea in context, where the problematic lies with the experience of the student, can allow him/her to see the representation as a tool to do what she/he wishes and can thus allow us to see how the tool can be a legitimate form of expression of their conception and a record of their actions in that process.

The Problem:

The current annual tuition(1987-88) for the Endowed College, Cornell University, is $12316. This is a result of annual tuition increases averaging 11.5% for the last 7 years. On the assumption that this annual rate of increase continues for the foreseeable future, the overall goal is to investigate expected future levels of tuition.

1. *Create a table which predicts the tuition levels for the next three years.*
2. *Find a method by which you could easily find the expected tuition 10, 50, or 100 years from now, and add these values to your table.*
3. *Create a graph showing this relationship.*

HOW MANY YEARS TO $250,000,000?

4. *A letter to the editor in the New York Times claims that with present rates of increase, Cornell's tuition will exceed 1/4 billion dollars within our lifetime. You would like to check the figures, and also be able to create a method by which you could predict the following milestones:*
   - *a) the year tuition first exceeds 100,000*
   - *b) the year tuition first exceeds 1,000,000*
   - *c) the year tuition first exceeds 10,000,000*
   - *d) Any dollar value you choose*

   Can you come up with a method that will 'undo' your original process, allowing you to use the level of tuition to calculate the number of years until that level is reached? If you have a problem, what is it? What kind of a function would you need to accomplish this task?

When faced with the problem of how to reverse or "undo the process", the student is likely to look at the tool used for doing the problem. We initially used this problem in a classroom situation where only calculators were available. Since we had asked students to keep records of their calculator keystrokes, an individual record might look something like this:

1.115	y^x	10	=	*	12316	=

Notice the y^x key is unusual, it is a binary key. Whereas our students have been successful in deriving inverses in previous (non-exponential) problems by "undoing" their calculator keystrokes, in this case they are stuck, since they have no way of "undoing" y^x. Although they will eventually find a way to solve their problem, typically by taking the log of both sides, this becomes for them an exercise in manipulation of logs, rather than an undoing of an operational process. We propose that:

1) teaching inverses on a calculator where procedures are carried out and recorded is a reasonable approach, and
2) inverses can be seen as undoing, a reversing of both the sequence and the operation of the keystroke record.

To assist in this process, we built in two features, a procedure builder and a binary log key. The procedure builder allows them to save a series of keystrokes under one calculator "button".

Thus in the example above, the given series of keystrokes could be saved as a procedure, and then used by entering first the number of years followed by a single keystroke that would carry out the set of operations above.

The binary log key, $\log_a x$ will allow them to directly undo the exponential operation a^x. In each operation, the calculator (as implemented in the software) will prompt the user to enter the desired base (any positive real $\neq 1$).

Although it is possible that we might have considered these implementations, without observing students, we think the procedures by which we made the decision to actually implement them are important:

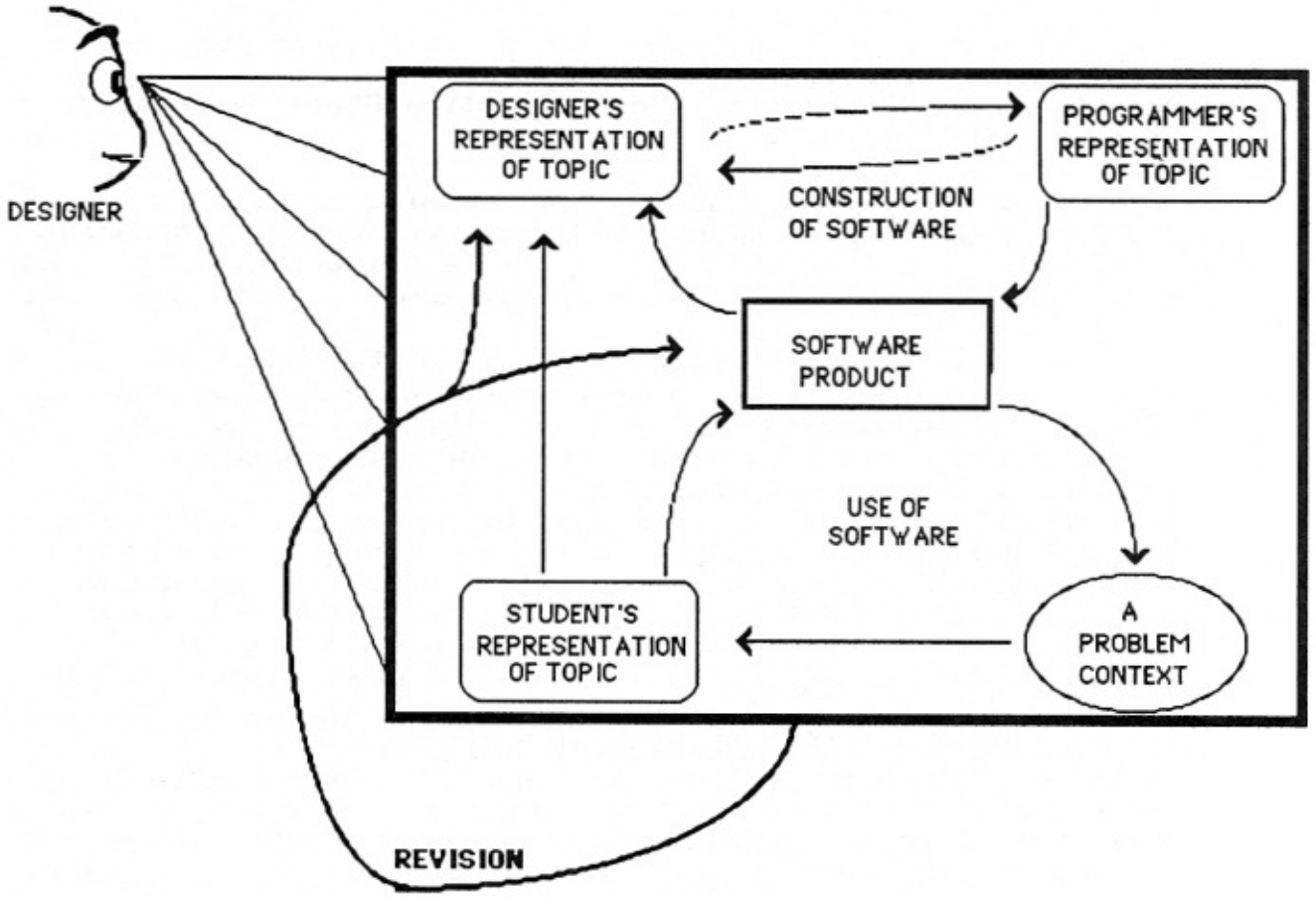
1) we observed the students preference for and facility with calculators
2) we suggested to the students that they start keeping records of their keystrokes as an alternative representation of a functional relationship.
3) This, in turn, suggested a way to think of and implement inverse operations.
4) we observed, again, a relatively easy and intuitive ability to carry out inverses by our students
5) when exponentials were used, we observed students reverting to rule-based log manipulations, due to their inability to undo the exponential key.
6) we decided to implement both the procedure builder and the binary log key.
7) we expect this interactive process to continue as students use the first version of the software, leading to ideas for redesign.

Conclusions:
We suggest that we need to distinguish three conceptual sources that jointly interact in software design:

1) the concepts of the designer in his/her intended version of a mathematical idea
2) the concepts as understood by the programmers who must translate a design document into an interface and a program
3) the developing concepts in the student's mind which are expressed and recorded through his/her interactions with the software.

Together these result in the implementation of a product, a set of iconic shapes and a set of operations that form the tool. If it is an effective product, it will allow for the genesis and diversity of student representations of their ideas to be expressed, acted upon, recorded, and reflected upon in their educational process.

We must remember, however, that it is still early in the game and that, so far, the effectiveness of computers in mathematical education is only beginning to be demonstrated. We suggest that their continued viability will be based on the recognition of: 1) the three influences mentioned above in product design, 2) student deviation as legitimate expression of mathematical ideas which we must interpret as input for further design, not inappropriate student effort, 3) transformations in our mathematical ideas resulting from back-talk from programmers and students. Design, as conceived in this fashion, will be more effective and our products more provocative.



Model for Student Centered Design Process