A Software Design Process to Facilitate the Teaching of Mathematics

Janelle Pollard and Roger Duke, University of Queensland, ITEE.

Abstract

Despite the existence of a large range of software to support the teaching of mathematics, most mathematical educational software makes only limited use of the potential user interfaces. In particular, students are typically restricted in how they can express their working and solutions. This paper looks at the issue of how to design mathematical educational software to help students solve problems in a natural unrestricted manner. The focus is on describing, through a case study, a process for designing software that mirrors the pencil and paper way students traditionally solve mathematical problems.

1: Introduction

Despite the existence of a large range of software to support the teaching of mathematics, most mathematical educational software makes use of very limited user interfaces. For example, many programs simply require students to input their answers to specific problems or present students little more than pre-prepared solutions.

The focus of this paper is to demonstrate one process for extracting possible interfaces that enables students to input their working in a natural format similar to what they would do with pencil and paper. The ultimate aim is to design software that adds to what a student can do with pencil and paper but without forcing upon the student a particular solution strategy and without taking from the student the need to be individually creative and inventive.

2: The case study

Students were asked to attempt to solve the following problem [3].

These counters show the digits 1 - 9 and two addition signs. They have been arranged to make a total of 1,368.

In what different ways can you arrange the counters make a total of 1,188?

There are three main reasons this problem was chosen. Firstly, the problem is conceptually simple and does not require particular mathematical skills or techniques. Secondly, it is open ended with a number of solutions possible. Thirdly, it can be approached in a number of valid ways.

3: Data collection

Seven students, each with different mathematical skill levels (from final year high school to honours degree) were asked to solve this problem with pencil and paper. They were allowed as much time as they needed, though all stopped working inside ten minutes. Six of the seven found at least one solution; four found more than one solution; none were able to say how many solutions existed.

Afterwards, each student was asked to explain their strategy; notes about their strategy were recorded, along with notes on any distinctive approaches or techniques.

4: Data analysis

Analysis of the students’ work can explain strategies used to solve the task and detect any errors in thinking [1][2]. Five common strategies for solving this particular task were identified:

- Swap two numbers of the original sum
- Find combinations for the ones or hundred columns
- Trial and error
- Subtract the answer from the example.
- Algebraic substitution for the number 1 to 9

(The first three strategies led to correct solutions, whereas the last two always seemed to end in dead ends.) The next step was to design an interface which allowed students to use as many of these problem solving strategies as possible. To do this, closer analysis of their working was required. Each worked solution was translated into a flow chart designed to capture the main steps in solving the problem, as performed by the student.

From the analysis of each student’s flowchart, common aspects were identified and five main interface components were identified. With these five components it is possible to capture the working of each of the students. The components were:

- the problem statement component where the original sum as presented in the problem was copied (this was usually the first step).
- a template component consisting of a 3 by 3 grid into which number could be inserted when searching for a solution,
- a combination/workspace component in which various combinations and sums of digits could be tested,
- a list of the numbers to keep track of which of the numbers 1 to 9 have been used so far,
- a summary component to record the attempts at a solution.
5: The combination/workspace component

Taking just the combination/workspace component as an example, three interfaces were designed. The first interface enabled students to input a number, the computer then returns a list of combinations of numbers summing to the supplied number. The second interface used a standard text area enables students to type in any numbers, letters and symbols found on a standard keyboard, starting in the top left-hand corner and working from left to right. Neither of these two designs allows for the way students actual used pencil and paper. Students often wrote downwards and even left to right when summing numbers. The third interface was created to facilitate this by enabling the user to click and type anywhere in the editor; if ‘enter’ is pressed the cursor moves down one line, relative to where they started typing.

6: Combining the various software interface components

We wish to combine the five main interface components in different ways so as to facilitate different learning strategies. One approach to help us overcome the common problem of discounting designs simply because they do not fit in with our implicit design beliefs is to insist we design at least one interface for each of the major categorise of software. There are several categorization systems for different types of programs [4][5][7][8]. In this case study we use the following categorization developed by Pollard [6]:

1. **Resource** electronic book
2. **Tool** computer as a (sophisticated) calculator
3. **Drill** given a question, input an answer and receive feedback
4. **Worksheet** given several questions, input answers and receive feedback
5. **Simulation** given a puzzle or virtual experiment, solve on the screen
6. **Discovery** presented with a scenario, manipulate the environment, check solutions or request more information

Interfaces for each of these categories were designed. The design which facilitates most of the five different components used by students is the discovery design. However, it is also the design which adds little to what is already facilitated by the humble pencil and paper. The tool design removes the tediousness of performing the addition and allows the student to focus solely on developing a strategy for solving the problem. The simulation design on the other hand provides guidance as to where errors have occurred by having numbers which add up to the wrong value turn a different colour. Hence a combination of some of these features and compromises between the extremes may develop a more educationally beneficial piece of software.

Having created these designs there remains one thing to do before considering implementing, and that is to ask the following two questions:

1. **Am I adding anything to the solving of the task by using a computer?** (What benefit is there for using a computer instead of any other resource?)
2. **Can I capture in the program at least the most common methods used to solve the task?** (It may not be possible to allow students to, say, draw a diagram on the screen. If so, is it still worthwhile designing software?)

7: Conclusions

Through the design process described, a series of interface designs were created which each look at the problem from different directions. The designs ranged from allowing the student to input no working through to facilitating all working expressed on paper by the students. The simple collection of students’ work suggested several diverse interfaces and suggested various strategies for solving the problem.

More important than the actual interfaces designed, is the design process itself. The system for defining the task, collecting student solutions, analysing the flow of student working, grouping like methods and components, designing main components and finally designing interfaces from all existing categories of software, is a design process which we believe can be used to design and justify educational software that facilitates natural, unforced solving of tasks on computers.

References