Revelations in the design of Educational Mathematical Software

Janelle Pollard and Roger Duke
School of ITEE, University of Queensland, Brisbane, Australia
janellep@itee.uq.edu.au and rduke@itee.uq.edu.au

Abstract

After extensive research into the design of educational mathematical software for use in secondary schools, it has become apparent that conventional software design processes do not produce software which satisfies the specific needs of individual teachers.
This paper relates the conduct of four case studies in which software products were designed for use in secondary education. Each product was designed as a partnership between a software developer and an Australian secondary school mathematics teacher; the products addressed four distinct mathematical topics. The educational philosophy of each teacher shaped the way the software package was designed and implemented in their classroom.
This paper focuses on the partnership design processes that were used to create these software packages, with a strong emphasis on analysing the interactions between the software engineer and teacher. From this, conclusions are drawn about the design processes needed to produce educational software products that will be accepted by the teacher and used in classrooms.

1.0 Introduction

A large survey of Victorian (Australia) schools concluded *"the most frequently reported software applications used in mathematics classrooms were generic - spreadsheets, word processors and internet browsers - rather than mathematics specific."* [4]
These are not new claims. Norton in 1999 reported that:

- *few secondary mathematics teachers used computers at least weekly;*
- *computers were considered equally or more effective than traditional instruction for doing calculations or providing basic skills practice;*
- *few teachers considered computers useful in developing conceptual understandings;*
- *no teachers used computers with less able senior mathematics students.* [5]

Our research, which has seen the conduct of four case studies and numerous interviews with Queensland Secondary school teachers, confirms this lack of acceptance into the classroom of computers as an aid to teaching mathematical concepts. The question we must ask ourselves is why are teachers generally not using the widely available software specifically designed to explore mathematical concepts? Several key reasons repeatedly emerge:

- Software is costly to purchase and hence schools are reluctant to spend money on something they are unsure of and that might be used only marginally, if at all;
- Schools often lack professional IT Staff. This makes it difficult to install software products and many teachers are therefore reluctant to invest the time required to set up the activities;
- Schools are often not technology rich, and lab time is a precious resource. (Even when there is the technology, many teachers are not aware of what resources they can access);
- Teachers want software that assists with the way they teach. It is very difficult for them to accept software that does not meet their specific teaching style.

It is this last reason that we wish to address explicitly in this paper.

Norton disputes some of these conclusions, particularly the teachers' claim of difficulty of access to resources: "*the coordinators consider this an excuse for: teachers' lack of knowledge about suitable software, concerns about the changing role of teachers, lack of time to plan computer-based mathematics learning, worries about not covering the syllabus and fears of computers.*" [5] Our experience shows that it is not just a case of knowledge about suitable software but in fact a lack of suitable software. Teachers will not use software for reasons like: different set out of worked examples, different expected working style, different way of wording questions, etc. as compared to their usual classroom teaching style.

We have conducted four case studies, with us acting as the software developer, aimed at analysing the process of co-developing educational mathematical software with secondary-level mathematics teachers. These case studies were designed to discover what common problems teachers face in integrating educational software into the classroom and how we, as software engineers, can overcome these problems and design software that meets teachers needs.

What has been discovered is that although many factors contribute to the low uptake of mathematical-specific education software in classrooms (see [1][2][6][7] for example), many of these factors can be overcome if we design educational software in teams, with the teacher and the software developer working together. Of course, for this to work, software engineers need to be trained more adequately for the task. This paper will look at the issues and implications of team-development of software for the four case studies. The research reported here is part of a larger project aimed at establishing a generic design architecture for the development of software to aid in the teaching of mathematical concepts.

2.0 Design Process

In all of the case studies the teachers were volunteers who had ideas about a software product which they wished to see implemented in their classroom. As a consequence the teachers were the driving force in the software's design and were responsible for all decisions regarding its educational content. In all of the case studies a common development model was used. This model was based on a phased approach and is listed in detail below.

The phases of the development

1. **Establishment of Topic.** This phase saw the teacher define the specific educational area.
2. **Definition of learning objectives and activities.** This was the phase where the teacher identified how they wished to teach this topic using the computer as a medium. They did this through defining the learning objectives for the software and by specifying the activities they would normally use to teach this topic. Worked examples were commonly provided by the teacher in this phase.
3. **Software requirements**. In this phase both the technical requirements for the software and its likely implementation were defined. Assumed previous student knowledge was identified. The way the software was to be utilised in the classroom was decided by the teacher. This covered such topics as to whether the software would be used by the class as a whole, individually or in small groups, who would direct the lessons, in how many lessons the software would be used, etc. User requirements relating to the students use of the software were specified by the teacher, along with specification of the computer equipment upon which the software was to be implemented.
4. **Software Design.** In this phase the worked examples provided by the teacher in phase 2 were converted into problems and activities that the software would then incorporate. The interface

was also defined, often using soft specification techniques such as storyboarding, to show how the software would deliver these problems and activities.

5. **Pre-production.** In this phase the software designer produced prototypes of the software to confirm that the configuration and presentation discussed were correctly portrayed. Any changes to the layout and content were made during this phase. The prototypes were in many forms, including sketches, story boarding, software demonstrations (with or without limited functionality), etc.
6. **Production**. This phase saw the implementation of the design and the refinement of the activities. This involved the conversion of the phase 5 prototypes into fully functional software.
7. **Installation.** In this phase the software was installed onto the school's computer network in preparation of use in the classroom lessons.
8. **Classroom implementation.** In this phase the software was executed in the classroom setting and feedback on the software's performance was sought from the teacher.

This evolutionary software development model was utilised in all of the case studies conducted.

3.0 The Case Studies

An overview of each of the four case studies will be briefly examined in the section. In Section 4 we will draw comparisons between the case studies with respect to the development phases. A summary of the general details of the case studies is given in Table 1. Screen shots and current versions of the software produced through the case studies are available at [8]. Whilst not all the case studies have progressed to phase 8, (i.e. full classroom implementation) all have completed the processes of constructing a design for a software package as a partnership between the teacher and software engineer. The main emphasis of this study is on the design process; however, relevant issues surrounding post-production are also discussed.

Table 1: General information about the case studies

	A	**B**	**C**	**D**
Name of Software Product	Exploring the relationships between line representations	Concept of converting units	Testing year 8 mathematics on-line	Looking at limits
Subject Area*	Patterns and algebra, chance and data	Measurement	Whole curriculum	Patterns and algebra
Status of Project	Classroom implementation	Implemented	Production phase	Production phase
Target year level	10	8	8	11
Average age of students	15	13	13	16
Teacher/s gender	female	female	male	male

* From the Queensland - Years 1 to 10 Draft syllabus document [10]

3.1 Case Study A

This case study looked at designing software to specifically teach year-10 students (14-16 years old) the relationship between the following representations of a straight line: data tables, ordered pairs, graphs and equations. Students had already seen all of these representations previously, however the teacher felt that many students did not explicitly link the four forms together, nor realize that they are simply different representations of the same information.

It was the teacher's belief that the change in medium, i.e. the use of computers, would assist in refocusing the students' attention and hence allow them to understand and visualize the relationships. The software was specifically targeted at under-performing students who were experiencing difficulties in this area. A screenshot of the software produced can be seen in Fig 1. This software was completed and successfully delivered to the teacher for implementation.

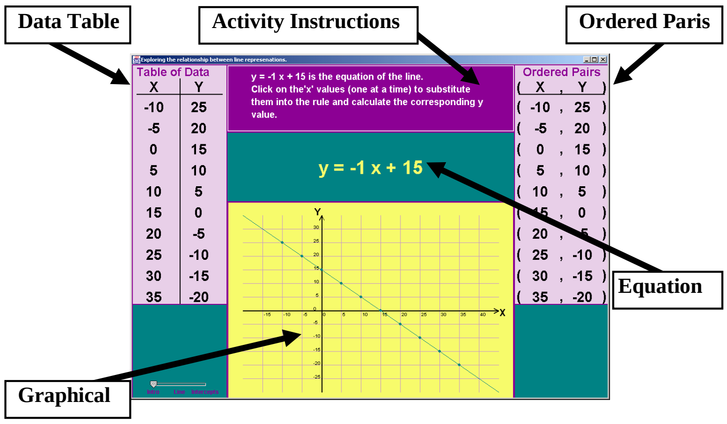


Table of Data

X	Y
-10	25
-5	20
0	15
5	10
10	5
15	0
20	-5
25	-10
30	-15
35	-20

y = -1 x + 15 is the equation of the line.
Click on the'x' values (one at a time) to substitute them into the rule and calculate the corresponding y value.

y = -1 x + 15

Ordered Pairs

(X , Y)
(-10 , 25)
(-5 , 20)
(0 , 15)
(5 , 10)
(10 , 5)
(15 , 0)
(20 , -5)
(25 , -10)
(30 , -15)
(35 , -20)

Figure 1: Screen shot of “Exploring the relationship between line representations”.

3.2 Case Study B

This case study looked at designing software to specifically teach year-8 students (12-14 years olds) the concept of how to convert between different units of measurement. This included teaching the base units of measurement and how to apply prefixes to these base units. The software aimed to reinforce the idea that converting units does not change the dimensions of an object; it also demonstrated the relationship between length, area and volume with respect to converting units.
This topic had been identified by a number of teachers as one which students often had difficulty understanding, both in that year level and in subsequent year levels. The teacher had attempted to use many other techniques to teach this topic but was still not satisfied and felt that software to target this specific issue would be of great assistance to both teachers and students.
The objectives were addressed through a series of introductory activities that involved introducing basic units, their size, prefixes and how they were used. Activities were then used which targeted length, area and volume and the relationship between them. A screenshot of the software produced can be seen in Fig 2. A full explanation of this completed case study can be found in [9].

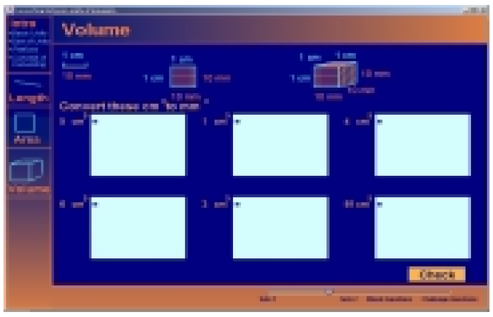
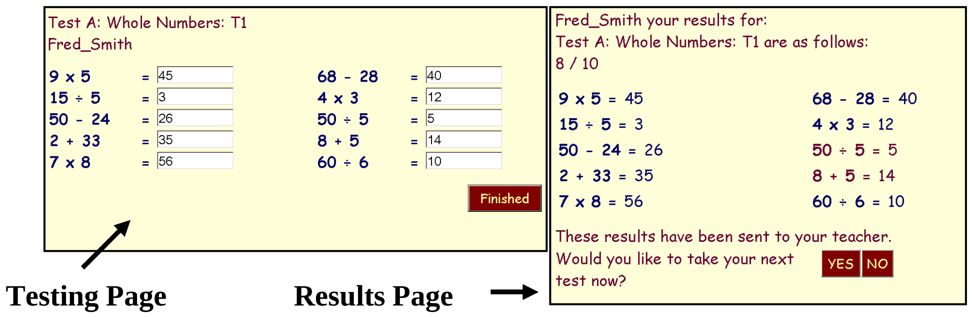


Figure 2: Screen shot of “Concept of Converting Units”

3.3 Case Study C

This case study looked at designing software to test year-8 students (12-14 years old) as they entered high school, in order to identify weak areas in their mathematical knowledge. The information gathered is to be used to sort the students into classes based upon these weaknesses.
This software is designed to test a number of different topic areas and provide informative feedback to both the teacher and student. The information given to the student and teacher differs in that the

student receives feedback on all questions answered whilst the teacher only receives the student's overall results. Unlike the first two case studies, it is not designed to specifically teach a subject area but rather to assess a student's knowledge level.
Currently this software is in the production phase and is awaiting the specific content of the test to be supplied by the teacher. Screen shots of the software can be seen in Fig 3.



Test A: Whole Numbers: T1
Fred_Smith

9 x 5	=	45	68 - 28	=	40
15 ÷ 5	=	3	4 x 3	=	12
50 - 24	=	26	50 ÷ 5	=	5
2 + 33	=	35	8 + 5	=	14
7 x 8	=	56	60 ÷ 6	=	10

Fred_Smith your results for:
Test A: Whole Numbers: T1 are as follows:
8 / 10

9 x 5 = 45	68 - 28 = 40
15 ÷ 5 = 3	4 x 3 = 12
50 - 24 = 26	50 ÷ 5 = 5
2 + 33 = 35	8 + 5 = 14
7 x 8 = 56	60 ÷ 6 = 10

These results have been sent to your teacher.
Would you like to take your next test now?

Figure 3: Screen shots of "Testing Year 8 Mathematics online".

3.4 Case Study D

This case study looked at designing software to specifically teach year 11 students (15-17 years old) the concept of limits. The software includes the ability to manipulate three representations of a function: equation, data table and graph, with several views of the graphical section available to assist the exploration of what happens as a function approaches a limit.
This was identified as a topic with which students struggled conceptually; it is one of the first "abstract" concepts taught to year 11 students. Students often have difficulty understanding and then applying what happens to a function as it approaches infinity, or a point of discontinuity, or a point at which the function is not defined. By using the computer to represent the function in different forms on the same screen, it is hoped that this will facilitate the students' ability to better visualise and understand what happens as a function approaches a limit. Currently this software is in the production phase and has taken its main design inspiration from Yunn Chao's work [3].

4.0 Comparison of the Four Case Studies

4.1 Differences in time spent to reach pre-production phase

While conducting the above case studies, information regarding the length of time required to progress through the different phases was recorded. Discrepancies were noticed between the lengths of time it took to reach the pre-production phase. These time differences are summarized in Fig 4.

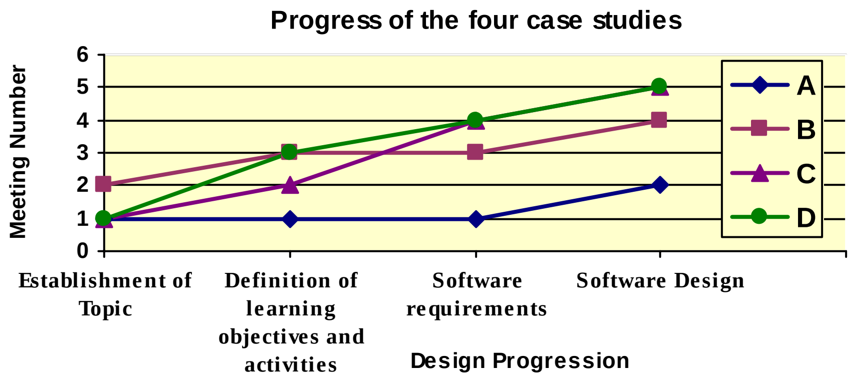


Figure 4: Rate of progress of the pre-production design phases

Discrepancies between the projects occurred mainly because of the different backgrounds of the software developer and teacher. The developer's personal knowledge and experience (or lack) of the mathematical topic was a significant factor in the speed of production in the first and second phases. Before the 'learning objectives and activities' could be transferred into 'software design' the software engineer needed to gain specific domain knowledge about the topic (i.e. how to perform the suggested activities). This was also true of the second phase - definition of learning objectives and activities - in order to guide the teacher to choose activities which would benefit from integration with the computer (and not just mimic what the teacher already does in a classroom without a computer). The developer had previously had personal experience tutoring the topic of case study A, which meant (see Fig 4) the project progressed significantly quicker than the other studies.

On the teacher's side of the partnership, the level of commitment to the production of the software affected the speed of progress. For example, whether the teacher had a firm idea of the topic, a clear objective for the software, a vision of how the software was to be used, and how best to teach the concept/technique, strongly determined the rate of production, (coupled, of course, with the speed with which the software engineer obtained personal knowledge and understanding of the topic in order to extract meaningful information during the teacher-developer discussions).

The teacher's knowledge of existing software became a significant issue in the software design phase, as it often limited their ideas as to what was possible. (On the other hand, screen shots of both existing and proposed educational software provided by the software engineer assisted in the expansion of ideas.)

4.2 Differences in the type of topic chosen

A lot of educational software focuses on teaching and re-enforcing techniques and conclusions. Statements such as: *"computers were considered equally or more effective than traditional instruction for doing calculations or providing basic skills practice; few teachers considered computers useful in developing conceptual understandings"* [5] would tend to suggest that teachers do not wish to use computers in their classrooms to assist in the teaching of concepts. Yet in the three case studies aimed at teaching (rather than evaluating/testing) students, all focused on teaching the concept; re-enforcing the techniques was a secondary outcome. The software was developed on the premise that straight techniques (as distinct from concepts) would be predominately taught away from the computers.

4.3 Differences in focus

Given the small sample size, clearly extrapolation is not appropriate. However it is interesting to note that both of the female teachers were predominately focused on designing the *content* of the software. The two male teachers, on the other hand, were very focused on the finer points of the design of the *software*. This correlated to the amount of regular computer use by the teachers: both male teachers made more use of computers to perform daily tasks.

4.4 Differences in reactions to examples of existing software

One technique utilised by the software designer to assist in software production was to show the teacher examples of existing software. This was intended to help stimulate ideas about how the computer could be used to tackle the educational problem. The technique met with different responses from teachers, as can be seen in Table 2.

Table 2: Teacher's response to examples

	A	B	C	D
Teachers responses to examples	Totally skipped example collection	Used examples to explain what was NOT wanted	Remained unmoved by examples and stuck firmly to original idea	Actively sort more and more examples and continually wanted to add new ideas into the design

For those who had a strong idea of the product they wanted to produce, as in case studies A and C, pre-existing software examples were of little benefit. In case D the teacher did not start off with a clear idea of how software could be used to teach the topic, so existing software was used to stimulate ideas of what the teacher would like to see in the design. Conversely, in case study B the examples of existing software was found to be useful in describing what aspect the teacher did not want to see in the software.

4.5 Differences in teachers' designing of content

The amount of guidance about the teaching objectives and activities of the software varied greatly between teachers. Not surprisingly, the firmer the teachers' personal conviction about what they wanted the students to get from using the software, and the way they wished to achieve this goal, the faster the project progressed. Interestingly, three of the teachers preferred to start at the learning objectives level and then construct the activities. However, in case study B the teacher rejected this approach and firmly started designing from set activities she specifically wanted to see incorporated into software; she then fleshed out the activities to achieve a more rounded understanding of the overall concept. (Table 3 contains a summary of teachers' design strategies.)

Table 3: Teacher's designs of content

	A	B	C	D
Teacher's pre-conceptions	Set objectives and set ideas about how to achieve the objective.	Set ideas about the activities to do but vague about the overall objectives.	Set objective but not clear on content.	Set objective but very vague on the way to teach the topic.

4.6 Evaluation of Design Techniques Utilised

A diverse range of soft specification techniques were used during the co-development process (See Table 4).

Table 4: Techniques used during development

	A	B	C	D
Existing lesson plans	-	✓	-	✓
Screen shots of existing software	-	✓	×	✓
Writing objective for the software	✓	×	×	✓
Collect sample working (from teacher)	✓	✓	×	×
Listing activities (with examples)	✓	✓	×	×
Sketches	✓	✓	✓	✓
Story boards of activities	✓	✓	✓	✓
Screen shot stories	✓	✓	✓	-
Prototype software demonstrations	✓	✓	✓	-

Key:	
-	**Technique not applied**
✓	**Technique was helpful**
×	**Technique was not helpful**

As can be seen from Table 4, most techniques were applied to each case study, but some proved unsuccessful in some circumstances. The teacher in case study B reacted negatively to being asked to write a sentence about the aims/objectives of the proposed software. However, this was not a problem, because alternative design strategies (like collecting sample working, listing activities, story boarding, etc.) lead to ideas being formulated about the overall learning outcomes.

The teacher in case study C also resisted the notion of writing the software's teaching objective, and gave only very sketchy examples to illustrate the scope of the activities. This, coupled with the teacher's reluctance to consider existing examples of software caused the stagnation of the project at the pre-production phase. This highlights a project's dependence on the teacher in the co-design process to provide expertise in matters of educational content. The software engineer can be expected to gain enough domain knowledge of the mathematical topic to actively be involved in discussions about content matters with the teacher, but it is unrealistic to expect software designers to create the content.

Case study D had the reverse issue. So many different ways of teaching the topic were looked at, that narrowing the design down to just a couple of approaches was a challenge; eventually this was achieved.

4.7 Issues with the installation phase

Although only two of the four case studies have reached the final installation phase, both studies that did had significant problems at this phase. The first problem in both case studies was the teacher's lack of knowledge of their school's computer systems. Even the technical staff tended to give very vague description of the systems. A consequence of this was the necessity in both cases to re-program code to allow for smaller screen resolutions than originally specified. The other common problem was the transfer of the software from the teacher's computer (a single computer) to the labs of computers (a network). The technical staff of three out of the four schools strongly resisted the concept of installing software onto the network systems fearing it would interfere with their network profiles and create more work for them. The teachers, understandably, tended to find this phase straining, and despite their enthusiasm about the software made comments about the amount of effort and rescheduling it took to run the lessons.

4.8 Acceptance of the software by other teachers

All meetings with the teachers were conducted in their staffrooms. As a consequence several other teachers looked over the projects and their reactions, whilst not surprising, were definitely interesting. Because each design was led by the instigating teacher's view of what educational aspects the software should tackle, a significant proportion of the other teachers in the staffroom did not see the software as something they would integrate into their teaching. Despite the software being co-developed by a teacher of their own staff, they usually did not wish to use it because:

- the worked examples were not set out in the format they use in their classroom,
- they wanted the activities in a different order,
- they would prefer the students could select rather than type the answer, etc.

In other words, they disputed the interface and the content; exactly the same reasons they gave for not using commercially available software in the first place. From Norton's and Copper's [6] work on factors influencing teachers' decisions to use or not use computers in their mathematical classrooms, other issues like the teacher's interpretation of the syllabi, the school's assessment model, the teacher's pedagogical beliefs about teaching mathematics, their knowledge of computers and the school's computer resources, etc. probably contributed to their reasons for rejecting the case study software for their own use.

4.9 Where does this leave us?

Our experience through the case studies draw us to the conclusion that few teachers are going to use software off the shelf, even if it is co-designed with teachers, unless the software is specially designed so as to be readily adapted to each teacher's specific requirements. The software produced in case studies has been strongly embraced by the cooperating teachers, and a few other teachers. But the conclusion is that 'all' teachers aren't going to be satisfied by any of the current off the shelf software packages. However, we can not keep producing software packages on a one to one bases with each teacher; this is unrealistic.
So what is the alternative? The reasons why teachers choose not to use a software package typically relates to very specific issues, like the way examples are set out. By changing the way software developers program the software we could make it possible for these 'issues' to be changed for each teacher. An example scenario: the developer comes into the staffroom and demonstrates their software package. Each teacher then tells the developer what they would have changed before they would use the package in their lessons. (That is teacher A only wants activity 3, Teacher B wants the whole package, but the wording of question 5 needs to be changed, etc.) The developer drops the changed 'components' into the package and gives each teacher their own version of the software. Hence we could keep the general structure and content of the package, but allow for the specifics to be easily tailored to the software needs of each teacher. To achieve this we need to change the way we design educational mathematical software.

5.0 Conclusions

Comparison of these four case studies has lead to the following conclusions:

- Both the software developer's knowledge of the topic area and the level of clarity in the teacher's ideas about the educational content of the software, directly contribute to the speed of development in the pre-production phases.
- Good communication between the teacher and software engineer is essential in the design process of educational software. To achieve this both parities must try to understand the other's contextual knowledge.
    - The use of screen shot examples of pre-existing software during the software design phase was the most successful technique used to facilitate the software engineers' understanding about the teacher's expectations of software, in general. This technique allowed the teacher to easily describe, in non-computer-technical terms, what they wanted from the software. (i.e. a teacher could point at interface widgets like radio buttons and discuss the object without requiring the computer domain specific knowledge of it's name, attributes etc.)
    - Supplying the software engineer with worked examples (i.e. questions and sample solutions) of mathematical problems within the chosen topic, was the best technique the teacher could adopt for facilitating the rapid growth of the software engineer's domain knowledge on the topic. Worked examples gave samples of questions, samples of solutions *and* samples of the expected setting out of the solutions.
- Contrary to some current literature, these case studies show that teachers *would* like to use computers to teach mathematical concepts. They were happy to teach mathematical techniques away from the computers, but strongly felt computer software could assist students to conceptualize difficult mathematical concepts.

Despite the case studies successes in gaining the instigating teacher's acceptance and satisfaction, it did not follow that the teacher's peers would embrace the software. Teachers have very individual teaching styles and they require tools that enhance the way in which they specifically teach. Given this, it is highly unlikely that a single piece of software, using current design methodologies, will satisfy even a majority of practising teachers. Schools do not operate the same as a business; you cannot force teachers to use a software package if they do not want to. They will do what they currently do with textbooks: use them only if they like them.

To overcome this issue it is our belief that the current software methodology needs to be changed so that we can produce software that is flexible and readily customised to meet the needs of each individual teacher. It is unrealistic to design a new software package for each teacher. However, the concept of component based software, i.e. software that, while maintaining a commonality of content, is made up of different components that can be put together in various ways in order to produce unique software solutions, would seem to be the way to progress. We are currently undertaking a larger project aimed at developing such a generic design architecture, in the hope that this will lead to a wider adoption of the computer as an aid to teaching mathematics, especially mathematical concepts.

References

[1] Barnes, M., Clarke, D. & Stephens, M. (1996). The impact of external assessment on teaching practice: Constraints on change in the classroom. In *Mathematics Education Research Group of Australasia* (MERGA). *Technology in Mathematics Education*, Australia, (pp. 65-71).

[2] Becker, H. J., Ravitz, J. L., & Wong, Y T. (1999). *Teacher and teacher-directed student use of computers and software. Teaching, learning and computing: 1998 national survey. Report #3.* Center for Research on Information Technology and Organizations, University of California, Irvine, and University of Minnesota.
webpage: www.crito.uci.edu/tlc/findings/ComputerUse/html/startpage.htm

[3] Chao, Y. (2002) "Approaching…but not equal to…": A Dynamic Visualization Tool for introducing the Concept of Limits to Pre-calculus Students. Final term paper for ED 299x Visualization in Learning, for Professor Roy Pea, School of Education, Sanford University, California. webpage: ldt.stanford.edu/~chaoyc/course/spring/Final%20term%20paper.doc

[4] Forgasz, H., Prince, N. (2001) Computers for secondary mathematics: Who uses them and how? In *Proceedings of the Australian Association for Research in Education* (AARE), Australia: Fremantle. (pp. 386-393).

[5] Norton, S. (1999). Secondary mathematics teachers' responses to computers and their beliefs about the role of computers in their teaching. In *Making the difference: Proceedings of the Twenty-second Annual Conference of the Mathematics Education Research Group of Australasia* (MERGA), Australia (pp. 404-410).

[6] Norton, S. & Cooper, T. (2001). Factors Influencing Computer Use in Mathematics Teaching in Secondary Schools. In *Proceedings of the Mathematics Education Research Group of Australasia* (MERGA). *Numeracy and Beyond*, Australia. (pp. 386-393).

[7] Norton, S. McRobbie, C. & Cooper, T. (2000) Exploring Secondary Mathematics Teachers' Reasons for Not Using Computers in Their Teaching: Five Case Studies. *Journal of Research on Computers in Education*, 33(1), (pp.87-107).

[8] Pollard, J. (2003) webpage: www.itee.uq.edu.au/~janellep

[9] Pollard, J. & Duke, R. (2003) Creating a Teacher and Software-Developer Partnership. Sixth International Conference on Computer Based Learning in Science, (CBLIS), Nicosia, Cyprus. (pp. 313-321).

[10] Queensland Studies Authority (2003) Years 1 to 10 Mathematics Syllabus (Draft July 2003).
webpage: www.qsa.qld.edu.au/yrs1_10/kla/mathematics/syllabus.html