What do you do when the technology is pulled away from under you?

Ian Johnston
School of Physics, The University of Sydney, Australia

Introduction

This is a cautionary tale about the dangers of using Information Technology in teaching, dangers which are only now becoming widely appreciated. It comes from physics, that being my subject, but I'm sure there must be similar stories within most other disciplines.

For me it began in 1988 when there was a very influential conference held at the University of North Carolina, with the simple title "Computers in Physics Instruction". While it is true that computers had been used in physics education for a long time before then, it is probably fair to say that a very large number of academics, myself included, got their first real taste of the possibilities on the new instructional methods at that conference, or from reading the proceedings[1].

The manifesto was loud and clear. What makes physics a difficult subject is its heavy reliance on analytical mathematics, but now that fast, cheap computers with good graphics were widely available, they could replace this mathematics with numerical computation, and present the results in pictorial form. You could teach material which used to be considered far beyond your students' grasp. The introduction of computers into physics teaching would change not only *how* the subject is taught, but also *what* is taught.

O brave new world, that has computers in it!

Fired with enthusiasm from that conference I persuaded my home department to introduce comprehensive courses in computational physics. We chose to work with a scheme know as the *Maryland University Project in Physics and Educational Technology (M.U.P.P.E.T.)*[2]. It was based on the philosophy that students should get experience in every stage of the problem solving process, which meant they had to do at least some of their own programming. The project authors chose to work in Pascal, because that was the programming language most Computer Science departments were teaching, and they used the proprietary system *Turbo Pascal*[3] because it was cheap and readily available. What *M.U.P.P.E.T.* contributed were a few well designed utilities to smooth the time-consuming and error-prone chores of data input, the setting up and drawing of graphs and the making of program direction choices. Thus the students could be in charge of their own learning, not the computer.

We set up two courses at second year level, taught in a microcomputer laboratory, dealing with quantum mechanics and electromagnetism. The structure of the teaching program was changed - from 4 hours lectures and 4 hours laboratory per week, to 3 hours lectures, 3 hours laboratory and 2 hours computer laboratory. From the start, this kind of computer teaching was well and truly "embedded" in the curriculum. We also established a third year course, covering Fourier

Transforms, and another in first year dealing with simple harmonic and chaotic motion of oscillating systems.

An important development occurred in the mid '90s with the publication of the *Consortium for Upper-level Physics Software (CUPS)* project[4]. This was nine sets of simulations in advanced fields like optics, electromagnetism, quantum mechanics etc. also written in *Turbo Pascal* and including an extensive set of utilities, based on, but far surpassing, *The M.U.P.P.E.T. Utilities*. The level of these simulations was, and indeed still is, much higher and more comprehensive than anything similar available for teachers. We adapted a course on optics to use this material.

So, within ten years of starting in this direction, an enormous amount of time and effort had been invested in these courses, both our own part (I'm sure anyone who has written teaching software can appreciate how much work would have gone into setting up the courses I have described) and even more so on the part of the *CUPS* consortium.

And then the problems started.

The first cloud on the horizon was the realization some years back that Computer Science departments were no longer teaching Pascal. The students coming to us, if they knew any programming at all, didn't know Pascal. We had always assumed that, provided we only asked students to modify programs, rather than write them from scratch, they could pick up enough programming knowledge for themselves. We certainly couldn't afford the time to teach it. But levels of dissatisfaction rose. Students increasingly resented being asked to learn a language they saw as archaic, and lacking the glamour of the new, user-friendly interfaces.

So we (partly) abandoned the idea that all students should program, and went to pre-written software. That was when the *CUPS* programs were brought in for the optics course. The classes do not look so very different, but now all the student time is devoted to physics, rather than to programming concerns. Many consider this a good thing. But nothing is ever that simple. When this course was introduced for the first time, errors were discovered in the accompanying book. When this happens, there is always a tendency for academics to retreat into an I-can-do-it-better-myself frame of mind. Even though the publishers agreed to correct those errors, *staff* acceptance of the course was compromised.

We still believed that students should do *some* programming in their undergraduate courses, if only to give them a marketable skill. We decided that the third year course should continue to require students to write code, but that it should be more up-to-date, object-oriented, event-driven code. Of course this re-awakened the argument about which is the most appropriate language for them to learn - an argument which always smacks of religious fundamentalism. In the end, the course was restructured retaining Pascal, but using the *Borland Delphi*[5] programming environment.

It seems to be working well, but a lot, lot more work had to be done to convert the teaching programs over to the new platform. Even though it is still the same basic language, translation will generate bugs, and what works on one platform will not necessarily suit another. Teaching programs must, of necessity, be robust and as bug-free as is humanly possible - otherwise they teach quite the wrong message.

It gets worse.

A real storm cloud blew up only this year. It had not sunk into our consciousness that Borland had ceased to support *Turbo Pascal* some years back. So far as we were concerned that didn't really matter. *Turbo Version 7.0* was still around, and that worked just fine. But didn't I say things are never simple? It was only when we came to upgrade some of our computers this year, to 300MHz, that we discovered, along with many all around the world, that *Turbo Pascal* won't run on these new machines! It's something to do with being too fast. Who knows? Who cares? Certainly not Borland. They've stopped supporting *Turbo*, and that's that. In the meantime what will become of our teaching software, and to the *CUPS* materials (among others), and all the time and effort that has been invested?

It so happens that a stop-gap solution is available. There is a patch that you can apply[6], and that will keep things working for a few years yet, presumably until your next computer upgrade. But it's a kludge, and that hardly seems a sensible basis on which to plan your teaching strategy. Sooner or later, we are all going to have to shift everything from *Turbo Pascal* to something else.

Sooner, rather than later, is what we've decided. After yet another soul-searching exercise, we're going to try our luck with *MATLAB*[7]. Even though *MATLAB* seems primarily designed as a kind of super calculator, it can be used for some high-level programming. So by learning to use it, students should gain some of the skills we have, in the past, considered important. Furthermore, in our university the Engineering and Mathematics departments seem keen on it, so at least we will fit in with the other computing experiences our students are gaining on campus.

But it means a lot of work. All those teaching materials that we have lovingly developed over the years have to be re-written. The physics and the mathematics is all there and doesn't have to be rediscovered but, as we all know they are not the problem. It's the programs in which they are embedded that spawn all the bugs, and cause all the angst. Rewriting them is going to be every bit as long and tedious a process as it was before.

Where to now?

Looking back over the past decade during which our computational physics courses have run, we can say that they have been successful, in that they have achieved most of the aims we originally set for them. They are expensive and time-consuming, but not significantly more so than some other forms of teaching. The running cost of the computer laboratory is comparable with, or slightly less than, that of ordinary laboratory teaching. The set-up costs are large (though not more so than an experimental laboratory), because writing a new computational physics course is a very big undertaking, and the cost has to be amortized over many years of use. We just won't be able to afford to teach this way if the courses have to be rewritten every time the technology changes. There *must* be some way of protecting the intellectual effort we put into our teaching materials from changes in the technology.

You see the same problem all over the place. At an international conference I attended this year there were lots of bright young faces, all enthusing over the new way of teaching physics - with Java applets. Their simulations look splendid and extremely professional, but what was the physics they were showing? Cannon balls dropping from the tower of Pisa!! Come on. We've been there and done that. Eleven years ago, at the North Carolina conference, we saw simulations of projectile motion. I can understand that those who wrote those simulations then might be daunted by the prospect of having to learn Java, or whatever is the next enthusiasm, and that they would be content for the next generation to take up the load. But does the physics have to start from scratch each time - with simulations of cannon balls?

Unless we can find an answer to that question, I suspect Information Technology may not have the bright future in teaching that we once thought.

References

1. Redish, E. F. and Risley, J. S. (Eds). (1988) *The Conference on Computers in Physics Instruction*. Addison-Wesley, Redwood Ca.
2. Wilson, J. M. and Redish, E. F. (1989) Using Computers in Teaching Physics, *Physics Today*, **42**(1), 34-41.
3. *Turbo Pascal*, copyright Borland International, 1983-9.
4. The particular part of the project we used was: Christian, W., Antonelli, A., Fischer, S., Giles, R. A., James, B. W. and Stoner, R. (1995) *Waves and Optics Simulations*, *The Consortium for Upper-Level Physics Software*. John Wiley and Sons, NY.
5. *Borland Delphi*, copyright Borland International, 1983-96.
6. A patch program which claims to fix the problem is available at: http://www.geocities.com/SiliconValley/Bay/9553/tpbug.htm
7. *MATLAB*, copyright The MathsWorks Inc., 1984-96.

Ian Johnston
School of Physics
The University of Sydney
NSW 2006
Australia
idj@physics.usyd.edu.au