Teaching Without Technology

Philip Machanick
Department of Computer Science
Rhodes Univerity
Grahamstown
p.machanick@ru.ac.za

ABSTRACT

Technology is touted as a solution to problems in education. But is it? I report here on experiences with dropping use of slides in lectures and returning to working on the board. The apparent result is more interactive, engaged classes. Unfortunately there are too many other variables to make the experiences here definitive. The purpose of this paper is to provoke discussion on whether technology is overused in teaching when the goals of improving student engagement and general effectiveness of learning can be met many ways. Technology is not necessarily bad, but making it the starting point risks locking out non-technological options.

Categories and Subject Descriptors

K.3.2 [**Computing Milieux**]: Computer and Information Science Education—*Computer Science Education*

Keywords

Teaching methods, learning, active learning

1. INTRODUCTION

A widespread presumption is that technology has to improve education. Physicist Richard Feynman used to speak of "cargo cult science" [4]. By this he meant things that have the appearance of science but lack the critical elements that differentiate science from superstition – such as the superstition of South Sea Islanders who made their impression of the trappings of technology in the hope that it would attract "cargo" – goods they had seen brought in during wartime by planes. Despite the appearances of airports and so on, no cargo arrived. As Feynman saw it, science has to stand up to critical evaluation, which means studying all evidence, not only that which supports your hypothesis.

As computer scientists, we want to believe that applying technology to teaching will improve it – but does it? In recent years, I have started backtracking from this position because I find that imposing technology between the teacher and the class creates a barrier. Worse, it creates an inflexibility that old-fashioned chalk-and-talk can easily beat. In this paper, I present some experiences. In the spirit of Feynman, I do not claim these as definitive but present them to challenge the conventional wisdom – to claim a scientific outcome requires a study for which the data is not yet available. The experiences I report are necessarily anecdotal because there are too many variables in the examples I use to find definitive causes for particular effects.

I start with some background – my personal experiences backed up by education research. From there I outline experiences with a current second-year computer architecture course, and end with some ideas to take forward.

SACLA 2014, 25-26 June, Port Elizabeth, South Africa
.

2. BACKGROUND

My experience of lecturing goes back to when overhead projectors were commonplace but not universally accepted because they required hand-written slides, which took time to prepare, and required neat handwriting to be legible. With the advent of the low-cost laser printer in the mid-1980s, I was an early adopter of copying laser-printed slides onto transparencies. Presenting like that looked more professional than writing with chalk, saved re-preparation in subsequent deliveries of courses and it was technology and technology is what we do.

Since presentation tools appeared, I have used a wide range. Aside from the ubiquitous PowerPoint, I have used PDFs, Apple's Keynote and even web pages to present. All of that of course takes significant time to prepare, and makes lecturing that much more like entertainment.

Yet, does all this really aid learning? Self-efficacy [1, 5] has long been recognised as a prerequisite for coping behaviour. It is true that a teaching method that may excite self-confidence by making the material seem easier may prompt a perception of self efficacy – but what happens when that perception is punctured by reality?

Also, there is evidence that it is not applying technology as such to education that makes a difference, but using more effective methods that works. For example, algorithm animators may seem to enhance learning but do not necessarily do so any more than effective paper-based methods [2]. In general such methods appear to give best results when they require engagement by students [6], though the exact details of how visualisation and animation tools actually aid education remains a topic for research [13].

Much educational research in computer science in the past has focused on cognitive models. The social construction model of learning changes the focus from *what* is learnt to *how* knowledge is created and shared. The social construction model implies an active approach to learning, as learners discover what it is to be a member of a particular community of practice [10]. One approach to achieving learning in the social construction model is *action learning*, which goes in cycles of planning, action, reflection and analysis, leading to re-evaluating the project and starting another cycle [9].

Putting all this together, the goal should be finding the most effective method for engaging the students and encouraging them to learn to work like practitioners in their field. Presenting neat slides with carefully-contrived animations is nowhere close to what they will ultimately do in their place of work, whether in applying their discipline or in an academic setting. Given that the students are a distance from where they need to be, there has to be a balance between pulling them out of their comfort zone and maintaining their confidence (their self efficacy).

2.1 The Start

In 2012, I started experimenting with something new. I had a new small Honours class in Computer Architecture and my favourite book [7] had become far too expensive, so I wrote extensive notes, over 100 pages. You might say this is not so new: people have been doing this since the quill pen. But what was different was I relegated the computer to the background. Not since PowerPoint and friends appeared have I used the computer as an accessory, where I could pull it out to show off a web page, to run a snippet of code or to highlight an example in the notes.

It was a liberating experience.

I could present at the natural pace of the class. If an example needed more explanation, I could elaborate. If they got it straight off, I could skip detail. If I found they were getting bored, I could add in something I hadn't planned. To me it appeared to work well, though since I have no baseline for comparison (I previously taught this course in a different country), the 2012 Honours class is more anecdote than evidence.

Some of this of course can be done with slides. But slides box you into a mindset where there is just so much material you have prepared in elegant style with animations. You get wrapped up into presenting and lose contact with your class. In the race to get through your carefully-prepared material you risk getting ahead of them. That is not so bad with a small Honours class but with larger undergraduate classes, I was starting to develop a concern that over-prepared lectures were not working.

Concurrently with the 2012 Honours class, I had a second-year (CS2) C++ course where the class really struggled – C++ is a large, complex language, and the aim of the course was to introduce low-level concepts that C++ inherits from C as well as the increased complexity of the C++ class and template system. Since this was a more introductory class, I had carefully-prepared slides using animations and graphics. Compared with the Honours class, I found it more difficult to pace lectures right and to ensure that the class had the core concepts straight. At Rhodes, undergraduate courses are run in relatively intensive mode, with a single module run on its own rather than concurrently with others. This C++ course was run over 4 weeks, making it difficult to give students enough time to absorb difficult background in time to do anything interesting. Using computer presentations was no help – with so much to get through, it was hard to avoid the temptation to go charging ahead even when the class was obviously not keeping up.

2.2 More Recently

In 2014, I inherited a CS2 computer architecture course. The previously-prescribed book [12], also a classic of its type, was becoming far too expensive – almost R800 when I last checked. So I decided to apply lessons from the Honours class. We had at the same time decided to replace the C++ course by a C course, so I integrated the two

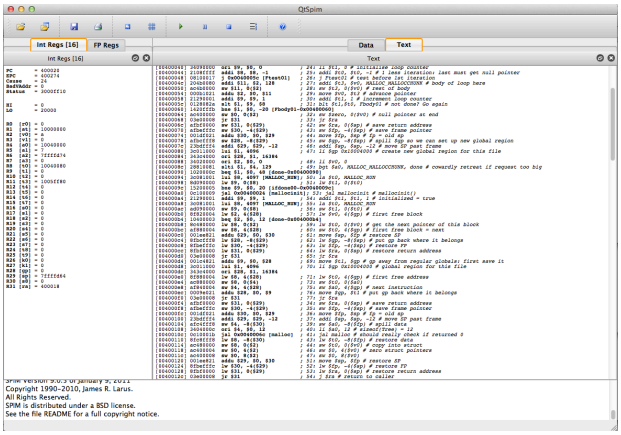

Figure 1: SPIM with tree example loaded

courses. The architecture course introduces the machine layer using C as pseudocode, and the C course picks up from there, dipping down to machine code as necessary to explain features. This too is not an original idea, though an available book using this approach [11] is also quite expensive, reinforcing my decision to write my own book.

In common with the previous version of the architecture course, I use the SPIM MIPS simulator [8] because the MIPS instruction set is one of the simplest in a real machine in wide use, and SPIM in its latest incarnation, implemented using the Qt toolkit [3], is reasonably portable, making it possible to run examples on most available equipment. SPIM as illustrated in figure 1 shows the loaded program and its machine code equivalent, and allows registers and memory to be inspected and altered. Memory on the whole is hard to inspect because the layout varies as non-zero values appear (regions that are all zero are compacted).

The presentation style is very old-fashioned. I work out at the start of the week what should go into the Wednesday afternoon prac, and work through the notes[1] to ensure we have covered enough ground to get the prac started. Every now and then I hit a point where demonstrating in SPIM becomes useful, and fire up the computer. The students have taken to the approach more than they realize. As noted in section 3, the students if asked would be happier with a more classic approach. However the results are generally good, and the class remained engaged.

Course content is biased towards MIPS assembly coding, with some cover of digital logic, number systems (including floating point) and performance. Since the goal is to elucidate high-level languages from the machine up, MIPS coding emphasises translation from C-like code to machine code, working through constructs from basic ALU operations via control structures to function calls and recursion. Data structures include arrays and C-style `structs`. While my material includes machine-level implementation of objects with inheritance, dispatch tables are one layer of complexity too much for the available time, so I did not go that far in the course. Nonetheless, we have covered a lot of ground, and the real test of the course will be how well the class copes with later courses that draw on understanding of lower-level concepts.

3. EXPERIENCES

[1] `http://homes.cs.ru.ac.za/philip/Courses/CS2-arch-C/`

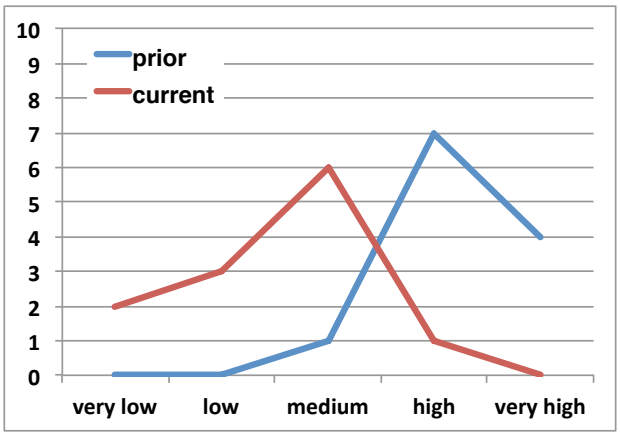

Figure 2: Self Efficacy: before and after – ***the vertical axis of all graphs is number of students***

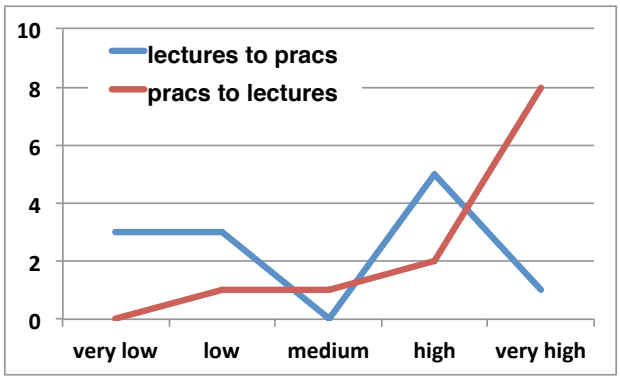

Figure 3: Value of lectures to pracs and vice-versa

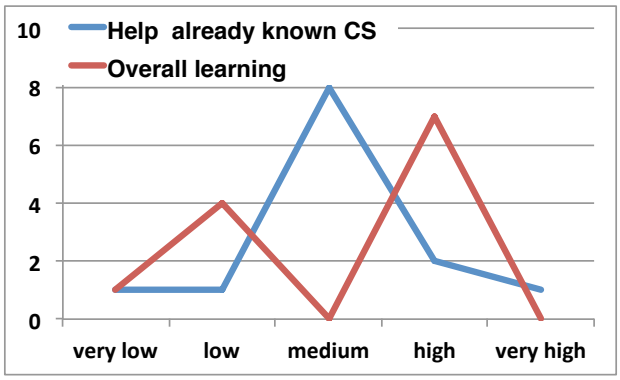

Figure 4: Perceptions of learning

We have a substantial number of CS2 repeats from 2013, a year with a higher than average failure rate, and the attitude seen last year when a large fraction of students gave up early and handed in nothing is not being repeated. Measuring 2104 CS2 against CS2 of 2013 therefore could be misleading, particularly as the group that arrived the following year has a totally different attitude that is unlikely to be related to any of my courses. Another variable is the introduction of a new first year in 2013 in which the Information Systems and Computer Science classes were split, creating space for a more focused first year.

In the 2014 CS2 class, the number of students failing to hand in is so low that I could give them individual attention. Since week 2, members of the class have voluntarily shown up for an extra weekend prac session. The first time, they organised this themselves. At the end of week 3, I joined in and the session that started at 2pm carried on for 4 hours.

In view of the uncontrollable variables, I have not attempted a formal study but rather present subjective impressions backed by an unscientific survey taken by 12 members of the class. Since the participants are self-selected with no controls for their representivity these responses can only be taken as a rough indication of the class's perceptions.

First, figure 2 is an indication of self-efficacy. The two data sets are the students' responses respectively on their confidence in their abilities before the course started, and at the time of the survey, about 3 weeks into the course. There is a clear shift of the peak of the distribution towards lower self-efficacy for the "current" data set. This shift is not surprising for a course that aimed to take students out of their comfort zone. Provided the shift does not discourage them in future courses, it is not necessarily a problem.

A perception students often have (no matter what you tell them) is that lectures are where most learning occurs. When lectures appear not to have the expected teaching effect they become disconcerted. There is plenty of evidence that learning is not passive; in a course where students are pushed towards active learning how do they react?

Figure 3 illustrates attitudes to the value of lectures with respect to preparation for practicals, versus the value of practicals for understanding lecture material. The distributions clearly differ. The students rate their practical sessions much higher for aiding understanding than the lectures. How does this reflect on the teaching strategy? My approach of building up to the next prac by the end of the Wednesday lecture may leave the class a bit off-balance by the time they reach the lab because they have the building blocks, but are expected to put them together themselves. The fact that they feel they are learning a lot from the pracs reflects the exact effect intended. Yet to the students, it appears that the lectures have not been sufficiently effective.

Finally, how well do the students think they are doing? Figure 4 illustrates responses to questions about how well the material learnt helps with areas of computer science they already know, and how well they are doing overall in learning from the course. Again, we see a split. The survey group is not really sure how much the course aids with other areas, with a peak at the midpoint of the options, while the level of perceived learning is bimodal, with some feeling they are learning a lot, and others rating overall learning low. The main benefits of the course will be to later topics including the follow-up C course and in their third year the operating systems and compilers courses. For this reason, it is not surprising there is some doubt as to the value of the course to other areas already covered. The bimodal distribution in students' perception of learning may be a consequence of taking the survey concurrently with the most difficult prac.

How does this actually translate to performance? Figure 5 is an extraordinary distribution and did not arise because the course was super-easy. The final prac required getting to grips with implementation of recursion, parameter passing, accessing components of a structured data type and loading a multi-file program into memory.

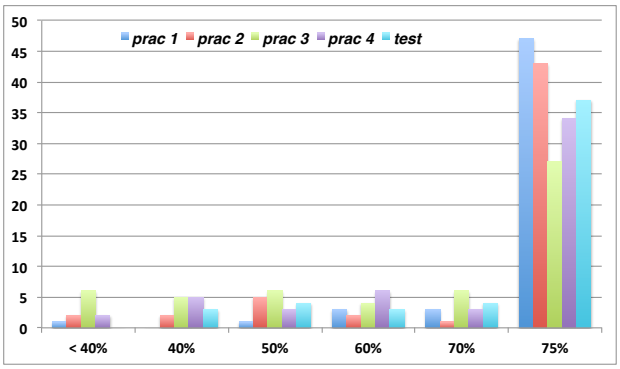

Figure 5: Class mark distribution – ***each horizontal axis label represents the minimum mark for that category, except "<40%"***

How much of this is down to the new approach? Given the big differences since 2013 – new first year the prior year, more engaged class, a different lecturer – drawing specific conclusions is not justifiable. However, dropping computer presentation can't have done much harm with such strong results. In 30 years of lecturing, I have never had a class with so many first class passes.

4. SUMMARY

With all the variables at play, drawing conclusions is not plausible – unless I am to move into cargo cult territory. Nonetheless experiences with the approach are useful to report for others to take up the challenge of measuring course improvement strategies without starting with technology. With more data points, we can actually do some science.

Given that we cannot easily compare the current second-year class with any other group, my experiences with this group at best provide a starting point for research in scenarios where test and control groups actually are comparable.

My argument here is for thinking about how to improve learning before we think about technology. If technology can aid in learning, good. But simply applying technology with no learning model in mind carries risks like giving students a false impression of learning, or leaving the class behind because it is too tempting to use up all the prepared material.

For me the most valuable outcome in writing my own book was being in command of the material in a way that is hard to achieve if you work with a book written by someone else. The fact that the book we abandoned was written by two industry leaders (Dave Patterson led the Berkely RISC project, inspiration for the SPARC architecture, and John Hennessy was the chief designer of the MIPS instruction set) was not a big cost to the students – neither of these very fine people is, after all, available to answer their questions.

5. REFERENCES

[1] A. Bandura. Self-efficacy: toward a unifying theory of behavioral change. *Psychological review*, 84(2):191, 1977.

[2] M. D. Byrne, R. Catrambone, and J. T. Stasko. Evaluating animations as student aids in learning computer algorithms. *Computers & education*, 33(4):253–278, 1999.

[3] M. Dalheimer. *Programming with Qt: Writing Portable GUI Applications on Unix and Win32*. O'Reilly, Köln, 2nd edition, 2002.

[4] R. P. Feynman. Cargo cult science. *Engineering and Science*, 37(7):10–13, 1974.

[5] V. Galpin, I. Sanders, H. Turner, and B. Venter. Computer self-efficacy, gender, and educational background in south africa. *IEEE Technology and Society Magazine*, 22(3):43–48, 2003.

[6] S. Grissom, M. F. McNally, and T. Naps. Algorithm visualization in cs education: comparing levels of student engagement. In *Proceedings of the 2003 ACM symposium on Software visualization*, pages 87–94. ACM, 2003.

[7] J. Hennessy and D. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kauffmann, San Francisco, CA, 5th edition, 2012.

[8] J. R. Larus. *SPIM S20: A MIPS R2000 simulator*. Center for Parallel Optimization, Computer Sciences Department, University of Wisconsin, 1990.

[9] P. Machanick. Peer assessment for action learning of data structures and algorithms. In *Proceedings of the 7th Australasian conference on Computing education-Volume 42*, pages 73–82. Australian Computer Society, Inc., 2005.

[10] P. Machanick. A social construction approach to computer science education. *Computer Science Education*, 17(1):1–20, 2007.

[11] Y. Patt and S. Patel. *Introduction to Computing Systems: From bits & gates to C & beyond*. McGraw-Hill, New York, NY, 2nd edition, 2004.

[12] D. Patterson and J. Hennessy. *Computer Organisation and Design: The Hardware/Software Interface*. Morgan Kauffmann, San Francisco, CA, 4 edition, 2011.

[13] J. Urquiza-Fuentes and J. Á. Velázquez-Iturbide. Toward the effective use of educational program animations: The roles of student's engagement and topic complexity. *Computers & Education*, 67:178–192, 2013.