Experience of Applying Bloom's Taxonomy in Three Courses

Philip Machanick
Department of Computer Science, University of the Witwatersrand
2050 Wits, South Africa
philip@cs.wits.ac.za

Abstract

A commonly accepted taxonomy of cognitive skills, Bloom's Taxonomy, puts analysis and synthesis near the top, with straightforward knowledge and comprehension at the bottom. Typical Computer Science curriculum discussion, though, usually focuses on which concepts to teach when, not on how to rank concepts or teaching methods for level of difficulty in terms of cognitive skills. This paper presents experiences of applying these ideas to teaching three very different Computer Science courses. These experiences suggest that taking Bloom's Taxonomy into account in course design is worthwhile.

1. Introduction

It is commonly accepted that the hierarchy of cognitive skills—as defined in Bloom's Taxonomy—has analysis and synthesis near the top, and straightforward knowledge and comprehension at the bottom [Bloom 1956]. In previous work [Machanick 1998*b*], I have presented a proposal to restructure the Computer Science curriculum, drawing on earlier work on abstraction-first learning [Machanick 1998*a*], to propose a curriculum which starts from lower-order cognitive skills, while working up to higher-order skills in later years.

This paper presents experiences with using the idea of ordering material according to required cognitive skills within particular courses. Experiences with three courses are presented. The first course is a Higher Diploma course, Data and Data Structures. The second is a third-year course, Algorithms and Artificial Intelligence. Finally, experiences with an Honours course, Computer Architecture, are presented.

The findings here are mostly anecdotal, and are intended as a first attempt at evaluating experience with the broader idea of applying Bloom's Taxonomy to curriculum design. On the whole, the experiences are positive, but a significant amount of work remains to be done to evaluate the idea properly.

The paper starts by considering some background, and goes on to describe each of the three courses in terms of what they set out to achieve. Some findings are then presented, and in conclusion, the value of the approach is summarized.

2. Background

Since the last major revision of the ACM/IEEE curriculum appeared in 1991 [ACM 1991], perhaps it's time there was a major review of our underlying assumptions. Curriculum '91 focused on defining *knowledge units* (KUs) at a relatively fine-grained level, so it left a lot of scope for curriculum experimentation. The current curriculum standards process underway under joint ACM and IEEE sponsorship aims to produce Curriculum 2001, with minor additions to Curriculum '91, and a set of standard curricula, for those unwilling to devise their own from KUs.

Against this historical approach to curriculum design, it is useful to reconsider whether simply deciding what *content* should be taught when is sufficient. Experience in the real world illustrates that computer software remains a problem area, and some have questioned whether a more "engineering"-like approach to Computer Science education is needed [Baber 1997]. Before such major upheavals in content are contemplated however, it is worth considering whether a reform in our model of what is "difficult", "introductory", and so on is necessary. Accordingly, this paper's starting point is to consider how Bloom's Taxonomy applies in the design of courses.

While earlier work [Machanick 1998*b*] has addressed the question of cognitive skills globally, in terms of overall curriculum for a degree, the approach in this paper is to question the curriculum development process at a different level: are we exercising the right cognitive skills within one course?

To fill in some detail, Bloom's Taxonomy [Bloom 1956] is widely recognized in school education as a basis for ordering material. Straight factual knowledge comes before comprehension. Application of knowledge comes next. Finally, analysis and synthesis (words which have a specific meaning in Bloom's work) are the most advanced cognitive skills.

In previous work [Machanick 1998*b*], Bloom's Taxonomy has been proposed as a basis for overall curriculum design, for the degree as a whole. In that model, with some input from other previous work on abstraction-first learning [Machanick 1998*a*], a 3-year curriculum would be structured as follows:

Year 1

- *use and measurement of applications*—know user interfaces and virtual machine concept; know major components of a computer; know functionality of a range of programs; know how user requirements are specified; know the purpose and use of user-level documentation; given a complexity result, measure a program's speed in a lab, and verify that a graph of run time matches the predicted speed
- *introduction to programming tools*—know the difference between source code, compiled code and libraries; know the steps in the program construction process; know the purpose of programming environments including editors, compilers and linkers; know the need for error checking and debugging; know the how correctness of software is specified and verified
- *basic network and operating system*—know the purpose of the operating system and networks; know the aspects of the virtual machine which the network and operating system provide; know virtual machines in more detail than before; know major operating system and networking concepts: processes, scheduling, resource management, memory hierarchy, protection, routing, the internet and internet-based tools
- *basic computer organization*—know the major components making up a computer; know the difference between machine code/assembly language and high-level languages; know the major microarchitecture components (registers, pipeline, buses, cache, main memory); know the role of I/O devices; know why the lower levels of the virtual machine are usually hidden

Year 2

- *basic program construction*—given a library, understand its interfaces and use its components to add to a given program; given an algorithm, construct a procedure or function to implement it; add a new function or procedure into an existing program and show that it works as specified; given a complexity result for an algorithm, prove it is correct; given two alternative algorithms, compare the known complexity results, and determine both theoretically and experimentally which is the better; given the specification of a data structure, use the data structure (implemented in a class library) in a program
- *formal languages and databases*—understand Turing machines, finite automata and pushdown automata; prove simple results for each model; construct simple programs in Turing machines, and simple recognizers in the other two (using supplied tools); understand the value of relations as a data representation; given unnormalized relations, perform normalization; understand the application of these concepts to databases; understand the general value and applicability of these formal approaches
- *advanced network and operating system*—implement simple examples of concurrency, given algorithms and interfaces to system calls; understand algorithms to implement queuing; understand issues in implementation of concurrency primitives and how they are used (locks, semaphores, critical regions, race conditions); understand resource management policies and algorithms including paging and scheduling; understand layered network models; understand major policy issues in networks including routing, congestion control, switches, routers and media
- *computer organization*—understand how logic circuits are building blocks of the major components; understand how to simplify logic and combine elements to create circuits; understand how design affects performance and how performance is measured; understand how assembly-language programs are written; relate assembly language on one hand to high-level languages and on the other to the hardware; understand how I/O devices interface to the hardware

Year 3

- *advanced programming and software architecture*—understand the architecture of an existing library or application and extend it; design a new relatively simple architecture (library or application): mainly project-oriented
- *compiler and DB projects*—use principles of formal language to implement small projects, to understand how compiler construction tools work (not only for parts of a compiler); do a database project; report on the value of the formal methods in these areas
- *network and operating system projects*—implement software using operating system calls, including use of pipes or other higher-level models of inter-application communication, multithreaded applications, use of low-level networking protocols such as UDP to implement higher-level protocols; simulation
- *advanced architecture*—design principles including instruction set architecture, performance impact of variations in design, overall system design, hardware-software interactions; code generation and assembly language

In this paper, the concepts are taken further to suggest how Bloom's Taxonomy can apply to teaching methodology within one course. The examples presented here are taken from a conventional curriculum, where the higher-level structure does not follow the above outline.

The approach, in essence, is to cover the factual content of the course quickly, in roughly half of the lectures, then go back over the material in tutorial and assignment mode. The idea is to exercise lower-level cognitive skills first, in lectures: there is little evidence that lectures in any case are an effective mode of teaching. Higher-level cognitive skills are then exercised by working through examples (comprehension), and solving problems (analysis). Synthesis is a higher-level skill than is demanded in any of these courses: in Bloom's model, it is a skill required in research and advanced courses.

3. The Three Courses

3.1 Introduction

The courses used as examples here have been presented in 1998 and 1999, using an approach based on Bloom's Taxonomy. The courses are at different levels, to illustrate how the concepts apply to students of different levels of preparedness.

The first course dealt with is a course called Data and Data Structures (DDS). This course was something of a stopgap course. My department runs a Higher Diploma in Computer Science, which is open to students with degrees but without any (or not much) Computer Science. The Higher Diploma is composed of undergraduate courses—most of the undergraduate curriculum, over a single year—taken at the same time as the courses are presented to undergraduate classes, which results in some ordering problems. The DDS course, as presented in 1999, was designed as a catch-up course, to prepare students to take on second-year material early in the year. It should probably have had a more descriptive name, but using an existing name saved us from having to implement a rule change (DDS is the name of one of our first-year courses).

The second course dealt with in this paper is Algorithms and Artificial Intelligence (AAI), a third-year course which has run for several years, though 1999 was the first year that I presented the course. This was a relatively conventional algorithms course, with some search techniques from AI thrown in.

The final one is an Honours Computer Architecture course, which I have presented for a number of years. Only this course can reasonably be compared with previous versions of the same course.

The remainder of this section provides more detail of each course, and how Bloom's Taxonomy was applied in presenting the material.

3.2 Data and Data Structures

The DDS course has already been described previously [Machanick 1999]. This section presents an update on previous results.

The course was designed as a bridging course, to bring students up to the level of being able to cope with a second-year course, Data Abstraction and Algorithms (DAA). DAA covered data abstraction, object-oriented concepts, algorithm analysis and data structure analysis. To make things difficult, DAA used C++. The Higher Diploma DDS course was designed to familiarize students with object-oriented concepts, libraries, working to an API and the general role of data structures and algorithms in Computer Science.

DDS was a necessary addition because the structure of the Higher Diploma was changed for 1999. Previously, some first-year courses and DAA were presented to the class separately from the undergraduate lectures, so prerequisite subjects could be presented first. For 1999, the decision was taken to move the Higher Diploma class, as far as possible, to the same lectures as the undergraduate classes, and DDS was the only course dedicated to the Higher Diploma programme. To make things more difficult, the DAA course was started early in the year, in the fifth week (after a short second-year database course).

DDS therefore presented a number of challenges. In addition, it was necessary to cover a lot of ground fast, because some of the students had no prior exposure to computers. Consequently a quick overview of operating systems, networks and programming languages was necessary. The group of students was highly mixed, from barely able to cope to able to keep up with the lecturer (if not get ahead).

Given the challenges of the course, it seemed worthwhile trying to do something out of the ordinary to make the course work.

The style of presentation of the course was to present factual content for the first three weeks in fast-paced lectures, with an increasing emphasis as the course progressed to tutorials, laboratories and classwork.

year	number at end	passed	failed	% passed
1995	11	6	5	55
1996	27	20	7	75
1997	16	13	3	81
1998	17	13	4	76
1999	21	18	3	86

Table 1. Statistics on HDipCS Success

The course can be considered to be a success because it replaced a much less demanding approach, without losing a large fraction of the class.

Table 1 contains a summary of outcomes for several years of the Higher Diploma.

3.3 Algorithms and Artificial Intelligence

The Algorithms and Artificial Intelligence (AAI) course has been running for a number of years but I had not presented it before 1999, so experiences here are not directly comparable to previous years.

The approach in this course again was to present material relatively factually first, then go back to deal with the material at a higher level of cognitive skill. Unlike for the DDS course, a text book [Brassard and Bratley 1996] had previously been prescribed for this course, and was used again in 1999. One of the previous lecturers had complained about this book, and it turned out to be a problem again.

The approach used was to present theoretical aspects of the course relatively factually, come back to do theorem-proving techniques, then apply the techniques to algorithms. This ordering is consistent with the Bloom’s Taxonomy-derived strategy. Results were reasonable, but the weakness of the book was exposed by this approach. Several approaches are described in the book, but not used to any degree of significance later for algorithms. As a result, it became difficult to tie the earlier and later parts of the course together. For example, the book introduces the technique of generalized induction without explaining it clearly. Consequently, I spent more time than I should have on explaining this concept, only to find that it wasn’t needed for anything later.

Although the differences in lecturers’ approaches makes it difficult to compare the course across years, it appears that any difference in outcome between the two years is difficult to assess because of problems with the text book. However, the course did not appear to be worse than would be expected, given these problems. So at least it is possible to conclude that the approach used was not harmful.

I will be running AAI again this year and this time around will have the benefit of hindsight in deciding which parts of the prescribed book to use.

3.4 Computer Architecture

The Honours Computer Architecture course has been run by the same lecturer since 1993, using two successive editions of the same book [Hennessy and Patterson 1990, 1996].

The course was run for the first time using the Bloom’s Taxonomy approach in 1999.

Since results are most comparable across courses in this instance, it is instructive to compare the course with its immediate predecessor in 1998.

In 1998, lectures were run in conventional style over 6 weeks. The course was run at the same time as another Honours course, as well as the Honours research report. After lectures ended, a week was available to finish assignments and to prepare for examinations. In 1999, the course was run over a similar period, but with very different use of time (for detail of contents, see Appendix):

- week 1—Chapters 1-3 of the notes (covering Chapters 1–4 of the prescribed book)
- week 2—Chapter 4 of the notes (covering Chapter 5 of the prescribed book)
- week 3—Chapter 5 of the notes (covering Chapters 6 and 7 of the prescribed book)
- week 4—Chapter 6 of the notes (covering Chapters 7 and 8 of the prescribed book)

In 1998, assignments were in the form of extension of tutorial examples, involving issues like timing of pipeline execution under varying conditions. There was not sufficient time to understand sophisticated tools to do hands-on work like simulations. In 1999, there was time to learn to use Simplescalar, a sophisticated architecture simulator. Two assignments

year	number in class	classwork average %	Classwork std. dev	exam average %	exam std. dev	overall average %	overall standard deviation
1998	8	69.7%	8.6	59.7%	14.0	63.0%	10.8
1999	17	73.4%	8.2	67.4%	11.0	69.4%	9.2

Table 2. Statistics on Honours Computer Architecture Outcomes

were done using Simplescalar. The first assignment was to explore the design space for memory hierarchy, the second, to measure aspects of instruction-level parallelism.

Other than the change of order and the adoption of more sophisticated assignments in 1999, the courses were substantially similar: both used the same book and the same lecture notes.

Table 2 summarizes outcomes of the 1998 and 1999 versions of the Computer Architecture course.

4. Findings

Results in the DDS and AAI courses are harder to evaluate than those of the Honours Architecture course, because there are too many variables as compared with other runs of similar courses.

DDS was new in 1999, and was run by a different lecturer in 2000, and there are too many differences in his style of presentation to compare the courses directly. Further, the DAA course to which DDS was designed to interface no longer existed in 2000, making it difficult to compare outcomes (in terms of preparation for the following course). However, as can be seen from Table 1, the Higher Diploma class did better overall than in previous years, despite the non-optimal ordering of topics.

AAI in its 1999 guise needed improvement in the choice of content (because of problems encountered in the prescribed book). I am running it again in the second half of 2000, and it will be interesting to see if it can be run significantly better, with better results than “conventional” runs of the course.

The Honours Computer Architecture course allows for some comparison in 1998 and 1999, even though there are some significant differences across the two years: the 1998 class was smaller, and had some very good students in it. Nonetheless, the course was given by the same lecturer in both years, and covered the same ground. The 1999 experiment was support for the revised approach, because a much more ambitious style of assignment was possible, and the class generally did well—despite the fact that the differences in the two groups of students argue for the 1998 class doing better.

As can be seen in Table 2, in 1998, the averages for classwork (assignments and tests), for the examination, and the overall average were lower than in 1999. It is also of interest to note the standard deviations were higher in 1998. Not only did the class do better in the 1999 course, but with less deviation from the average.

There are two downsides in general to the Bloom’s Taxonomy-based approach: students are uncomfortable at the start of the course because it is going so fast, and preparation has to be geared to the fast initial pace. For both reasons, the approach is probably best not tried by an inexperienced lecturer, who cannot maintain the confidence of the class, and keep up with the required rate of preparation. On the positive side, lectures are over sooner, leaving more time to think up creative classwork in the later stages of a course presented in this style.

5. Conclusions

Results from the DDS and AAI courses suggest that the Bloom’s Taxonomy-based approach works at least as well as conventional approaches. The DDS case is slightly stronger, as the AAI course had problems relating to good use of the text book, which could have obscured any advantage of the new approach. The DDS course can only be compared against relatively ambitious goals, and not against another version of the same course, and so is not an entirely satisfactory experiment. Unfortunately, changes in our undergraduate curriculum preclude further experimentation with this specific course.

The Honours Computer Architecture course provides the strongest case for the Bloom’s Taxonomy-derived method. In 1999, despite having a larger and weaker class, more sophisticated assignments were done, and the overall results of the class were better.

For 2000, the Honours Computer Architecture course is being run again along the same lines, but with the lectures at an even faster pace. The aim is to conclude the lectures in three weeks, rather than four, and to spend the second half of the course entirely on tutorials, assignments and tests. A similar approach is planned for the third-year AAI course in the second half of 2000.

Overall, the approach seems worth taking further, and trying with other courses. The problem identified in Section 4, of the approach being difficult for inexperienced lecturers, is also worth investigating. Perhaps if course material were suitably organized (particularly text books written in a suitable style), the approach would become more accessible.

Acknowledgments

Scott Hazelhurst and Ian Sanders proofread the paper.

References

[ACM 1991] A Summary of the ACM/IEEE-CS Joint Curriculum Task Force Report: Computing Curricula 1991, *Comm. ACM*, vol. 34 no. 6 June 1991, pp 68–84.

[Baber 1997] Robert L. Baber. CS Education and an Engineering Approach to Software Development, *Proceedings of the 27th Southern African Computer Lecturer's Association Conference*, Wilderness.

[Bloom 1956] Benjamin S Bloom (ed.). *Taxonomy of Educational Objectives: Book 1 Cognitive Domain*, Longman, London, 1956.

[Brassard and Bratley 1996] G Brassard and P Bratley. *Fundamentals of Algorithmics*, Prentice Hall, Englewood Cliffs, NJ, 1996.

[Hennessy and Patterson 1990] JL Hennessy and DA Patterson. *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Mateo, 1990.

[Hennessy and Patterson 1996] JL Hennessy and DA Patterson. *Computer Architecture: A Quantitative Approach* (2nd edition), Morgan Kaufman, San Francisco, 1996.

[Machanick 1998*a*] P Machanick. The Abstraction-First Approach to Data Abstraction and Algorithms, *Computers & Education*, vol. 31 no. 2, September 1998, pp. 135-150.

[Machanick 1998*b*] The Skills Hierarchy and Curriculum, *Proc. SAICSIT '98*, Gordon's Bay, South Africa, November 1998, pp 54-62

[Machanick 1999] Teaching Programming Backwards, Proc. *Southern African Computer Lecturers' Association Conference*, Golden Gate, June 1999, pp 69-73

Appendix: Computer Architecture Contents

- Instruction Set Principles: RISC vs. CISC
- broad classification
- Challenges for Pipeline Designers
- Causes of Pipeline Stalls
- Non-Uniform Instructions
- Techniques for Instruction-Level Parallelism
- Limits on Instruction-Level Parallelism
- Trends: Learning Curves and Paradigm Shifts
- Further Reading

Chapter 4 Memory-Hierarchy Design
- Introduction
- Hit and Miss
- Caches
- Main Memory
- Trends: Learning Curves and Paradigm Shifts
- Alternative Schemes
- Introduction
- Direct Rambus
- RAMpage
- Further Reading

Chapter 5 Storage Systems and Networks
- Introduction
- The Internal Interconnect: Buses
- I/O Performance Measures
- RAID
- Operating System Issues for I/O and Networks
- A Simple Network
- The Interconnect: Media and Switches
- Practical Issues for Networks
- Bandwidth vs. Latency
- Trends: Learning Curves and Paradigm Shifts
- Alternative Schemes
- Introduction
- Scalable Architecture for Video on Demand
- Disk Delay Lines for Scalable Transaction-Based Systems
- Further Reading

Chapter 6 Interconnects and Multiprocessor Systems
- Introduction
- Types of Multiprocessor
- Workload Types
- Shared-Memory Multiprocessors (Usually SMP)
- Distributed Shared-Memory (DSM)
- Synchronization and Memory Consistency
- Crosscutting Issues
- Trends: Learning Curves and Paradigm Shifts
- Alternative Schemes
- Further Reading
- Exercises