Experiences with Active and Collaborative Learning

Tom Briggs
Department of Computer Science
Shippensburg University
thb@ship.edu

March 26, 2005

1 Introduction

Active and collaborative learning allows students to work in teams to accomplish some specific goal, such as solving a problem or writing a piece of code. Active learning is a common technique used to improve students comprehension and retention of material [1], [6]. The most common application of active learning described in the CS education literature is in introductory programming courses [3], [9].

In these introductory courses, techniques such code review, debugging sessions, problem solving, and peer collaboration are commonly used to show improvement in the comprehension and long-term retention rates of material. Research shows that students show a higher rate of retention and mastery of the material and lower drop out rates [3]. Another study shows that active learning promotes as much as a 70% increase in the amount of long-term retention of knowledge. [6]

Students of most disciplines demonstrate better mastery of the material when they are able to interact with it but research in learning styles assessment suggests that it is especially true for computer science students [8].

1.1 Felder-Silverman Index of Learning Styles

The Felder-Silverman Index of Learning Styles breaks learning styles into four groups [7]:

- Active and Reflective
- Sensing and Intuitive
- Visual and Verbal
- Sequential and Global

Active learners prefer an environment that enables them to learn by using the knowledge such as writing programs or discussing material with their peers. Reflective learners prefer

an environment than enables them to cogitate over the material. Sensing learners prefer learning facts and concepts, intuitive learners prefer learning possibilities, applications, and relationships. Visual learners prefer learning from material they can see: charts, figures, and demonstrations. Verbal learners prefer words, either spoken or written. Sequential learners follow material in a step-by-step sequence. Global learners tend to learn by putting material into a global context and seeing how the material relates, and then they will "get it." [7].

In [8], Thomas, Ratcliffe, et.al, report the results of administration of the Felder-Silverman survey to 107 students in an introductory CS course. The results show a near even split for sensing/intutitive and sequential/global. There was a tendency towards active learning (55%) over reflective (45%). The strongest difference was visual (83%) vs. verbal (17%). The results of this survey strongly suggest that CS students are more visual learners.

The results of the CS student previous survey adds a quantitative dimension to the argument. If CS students are more visual learners, then one of the most effective learning environments is one where they are able to interact with the material in some fashion, in other words, an active learning environment.

1.2 Active Learning Techniques

Active learning does not require a large amount of preparatory work. Felder proposes the following simple approach to incorporating active learning in a class[2].First, ask the students to interact with the material, for example, ask them to solve part of a problem, derive a piece of code, sketch out a proof. Next, ask them to break up into groups, and tell them how long they have to solve the problem. Finally, after the time is expired, call on a few groups to share their solutions .

The important part of this strategy is that incorporating this break in the flow of the class allows students to refocus their attention. Students are able to interact with their peers and try out their conceptual view of the material and get immediate feedback on their understanding of the material. Finally, pairing with peers helps academically weak students begin to comprehend. Felder suggests that even a five minute active period during a 50 minute lecture can change the dynamic of the entire lecture and keep students better engaged in the material.

2 Active Learning in CS Courses

The two courses presented here courses were taught with an emphasis on active environments. The first was Operating Systems, a four credit introduction to operating systems for juniors and seniors. The second was Computer Organization, a four credit two hundred level introduction to assembly language programming, computer architecture, pipe-lining and other ILP techniques, and external I/O devices for sophomores. The courses were taught Spring and Fall 2004, respectively. There were formal and informal active techniques used in these courses.

2.1 Operating Systems

The formal techniques in this course took the form of in-class worksheets that were passed out at the beginning of the class. For example, a worksheet was developed to support the influence of I/O processing on the speed of a machine. The worksheet challenged students expectations by claiming that two machines, one with a processor 10 times faster than another, would actually require the same amount of time to complete a set of jobs. The students were led through each section of the worksheet, which included places were students worked with partners to "fill in the blanks" and then used the given equations to derive the results. At the conclusion of the worksheet, discussed how the theoretical results connect with modern systems.

Throughout the semester, the course turned to informal and often ad-hoc active learning sessions. In many cases, during particularly abstract material, the pace of the lecture would be broken up, and a practical example would be introduced. For example, during an otherwise abstract discussion of file I/O, a quick in-class demonstration of the UNIX `mmap` system call was introduced through an instructor lead demonstration. Students were then asked to theorize if the traditional file processing loop or the `mmap` method would be faster / more efficient, and to justify their position. Students were also asked to enter the demonstration code on their own and to create a traditional program that read the same file, and finally to compute the time each required. Students were also introduced to the UNIX `truss` command which reports the interaction between their compiled programs and the kernel. Students used to `truss` on their compiled C programs to determine how the C compiler and system libraries actually implemented their instructions. Students perspective on I/O was changed in a way that a straight lecture would not have been able to do.

Another example of an informal technique used in this course happened while presenting material on multi-threading and context switching. The material in the book described, in an abstract way, how two different operating systems handled process creation and context / thread switching. The students were asked to speculate which system's implementation would be faster. A class-wide discussion allowed students to take this general question and develop a sound hypothesis. Their hypothesis was that Microsoft Windows would handle process creation more efficiently than the Linux system, but that Linux would handle context switches more efficiently. The class was lead through the definition of a test for the hypothesis. Students were assigned homework to implement the tests and collect results for the next class.

When the class reconvened, the results were, predictably, varied. Students came to different results and conclusions, mostly because they compared their personal PC's running Windows to shared servers running at different speeds with different operating systems such as Linux and Solaris. Students were then grouped into small groups (5 to 6 students) for the purpose of consensus building. Each group was given no more than 10 minutes to propose one theory and explain away the initial differences of opinions and results.

Exercises such as this enable students to test their comprehension of material, and exercise their past experiences. Students moved into what Bloom's Taxonomy describes as Evaluation and Synthesis, the two most advanced levels of learning [4].

2.1.1 Independent Active Learning

Active learning is not limited solely to the classroom. Students were assigned a semester project that asked them to develop a Linux kernel module. Students were given a set of minimum criteria their implementation had to meet and were given complete freedom to choose how they would meet those specifications.

The kernel module they were asked to write was relatively simple. Students were linked to an on-line tutorial that gave instructions for writing a kernel module. The project specifications included a list of files they would most likely need to use to complete the project.

This type of project is an active learning environment because it requires students to be involved with the material in a hands-on way. The project required the application of many of the techniques taught throughout the class (e.g. use of semaphores inside their kernel module). Students, working in teams, became their own support groups, tutoring and supporting each other throughout the project and other parts of the course.

2.2 Computer Organization

The techniques developed for the Operating Systems course, such as class-wide open discussions, and hypothesis building worked well for upper-class students. Most of the students already completed many theoretical courses, such as Algorithms, and Simulations, and thus they had experience developing and testing hypotheses. Computer Organization was a sophomore class, and demanded a different set of techniques.

The computer organization course used fifteen worksheets [1]. These worksheets were more structured than those for the Operating Systems course. The content steered the students more and required less original thinking.

Effects of Pipelining on Sun Ultra SPARC-III

1. What do you expect will be different with respect to run-time, code size, and cycles per instruction when loop-unrolling is on or off?
2. Copy the source code (loop.c, loop2.c, Makefile) from my directory:
   `/home/tbriggs/csc220/pipe`
   into a working in your home directory.
3. Use the command `make OPT=A` to build the software.
4. Run the code through cpu-track:
   `cputrack -c pic0=Cycle_cnt,pic1=Instr_cnt ./loop`
5. Compute and record the CPI for the machine
6. Rebuild the software with the command `make OPT=B`
7. Rebuild and re-run the software as in steps 2 and 3.
8. Compare the results, and speculate wether loop-unrolling was on in the first example, and if it was on in the second. What were the reasons for your speculation?

Figure 1: Example of a Computer Organization worksheet

[1] Available at `http://www.ship.edu/~thb/csc220`

2.2.1 Simulators and Counters

Simulators and hardware counters are ideal environments for students to support their theoretical knowledge with concrete examples. In the Computer Organization course, there were two primary packages used. The simulator, SimpleScalar, is a MIPS simulator, and comes with a version of the GNU C compiler that cross-compiles code for the simulator [5]. Students can compile C code using the cross-compiler, execute that code in the simulator, and collect various performance statistics such as cache performance, cycle and instruction counts, and CPU stalls.

Hardware counters are available in most modern processors. With the addition of a tool to configure the processor's status registers, software can monitor the counters during the execution of a program and report statistics from an actual machine. The Sun `SPARC-III` architecture includes support for hardware counters that allows an application, such as Sun's `cputrack` program, to compute the number of instructions and clock cycles a particular program used during its execution.

Simulators and hardware counters create an ideal environment for active learning. Students can experiment with various configurations of the compiler and simulator and compare results. For example, a worksheet on CPU caches asked students to run programs through SimpleScalar's cache simulator, and identify the best configuration of cache they could.

Using real programs, running on a real machine, with real statistics from hardware counters gave students a set of concrete experiences to help solidify their abstract understanding of problems such as pipeline performance and the effects of context switching and background processing on their application. In many cases, there were several inconsistent results students found throughout the semester. For example, by turning on certain compiler optimizations, some programs showed reduced run-time but degraded cycles per instruction. Students were asked to develop explanations for these inconsistencies, and as a class, they were proposed and discussed.

3 Conclusions

Active learning environments are an effective style of teaching. Research suggests that active learning is especially effective for CS students who tend to be visual/intuitive learners.

Active learning was used in two courses for students at different class standings. In each case, formal and informal active methods were employed to engage the students in otherwise impenetrable material. Techniques such as frequent in-class problem solving, worksheets and discussions were often used in place of lectures.

Using an active approach helped students move up from the lower levels of Bloom's taxonomy (simple knowledge and comprehension) into the highest levels of comprehension (analysis, synthesis, and evaluation). Students attained a greater level of understanding of the material because they had the opportunity to interact with it. Students were guided through the development of a hypothesis, testing their particular hypothesis, and explaining how the results support or refute their theory. This process helps students test their knowledge and understanding of a problem.

There are no quantitative results that show that these techniques had a significant impact

on student outcomes. There is strong anecdotal evidence of the success of these methods. Student exam scores were strong given the difficulty of the material. Challenging students to use the scientific method to develop and test hypotheses changed the dynamics of the class, and helped even weaker students actively participate and engage in the material. Student evaluations of both the OS and Computer Organization were very favorable. A Shippensburg University standard student evaluation instrument includes a question about the most positive aspects of the course. Over 60% of those who responded listed the worksheets and class discussions as the most positive aspects of both the Operating Systems and Computer Organizations course.

References

[1] Owen Astrachan, *Concrete teaching: hooks and props as instructional technology*, ITiCSE '98: Proceedings of the 6th annual conference on the teaching of computing and the 3rd annual conference on Integrating technology into computer science education, ACM Press, 1998, pp. 21–24.

[2] R. M. Felder and R. Brent, *Learning by doing*, Chem. Engr. Education **37** (2003), no. 4, 282–283.

[3] Scott Grissom and Mark J. Van Gorp, *A practical approach to integrating active and collaborative learning into the introductory computer science curriculum*, Proceedings of the seventh annual consortium on Computing in small colleges midwestern conference, Consortium for Computing Sciences in Colleges, 2000, pp. 95–100.

[4] Lewis E. Hitchner, Judith Gersting, Peter B. Henderson, Philip Machanick, and Yale N. Patt, *Programming early considered harmful*, SIGCSE '01: Proceedings of the thirty-second SIGCSE technical symposium on Computer Science Education, ACM Press, 2001, pp. 402–403.

[5] SimpleScalar LLC, *Simplescalar 3.0*.

[6] Jeffrey J. McConnell, *Active learning and its use in computer science*, ITiCSE '96: Proceedings of the 1st conference on Integrating technology into computer science education, ACM Press, 1996, pp. 52–54.

[7] K. Silverman R. Felder, *Index of learning stylels*, World Wide Web., February 2005.

[8] Lynda Thomas, Mark Ratcliffe, John Woodbury, and Emma Jarman, *Learning styles and performance in the introductory programming sequence*, SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, ACM Press, 2002, pp. 33–37.

[9] Henry M. Walker, *Collaborative learning: a case study for cs1 at grinnell college and austin*, SIGCSE '97: Proceedings of the twenty-eighth SIGCSE technical symposium on Computer science education, ACM Press, 1997, pp. 209–213.