# Techniques for Active Learning in CS Courses

**Abstract**

Studies show that active learning promotes improved long-term retention of course material in students. Research suggests that CS students tend to have learning styles that make an active environment almost critical to their successful mastery of material. This paper demonstrates the effects of a novel lab experience in an objects-first Computer Science 2 course. Our lab is a novel departure from traditional lab based courses, in that it promotes student self-learning and interleaved into the class lectures. We show through assessment comparison that an active learning improves student grades, retention, comprehension, and satisfaction with the course.

## 1 INTRODUCTION

Active learning is a common technique used to improve students comprehension and retention of material [1], [4]. The most common application of active learning described in the CS education literature is in introductory programming courses [3], [7]. 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 [4]. Students of most disciplines demonstrate better mastery of the material when they are able to interact with it. Research in learning styles assessment suggests that it is especially true for computer science students [6].

### 1.1 Felder-Silverman Index of Learning Styles

The Felder-Silverman Index of Learning Styles breaks learning styles into four groups [5]:

- 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." [5].

In [6], 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 the survey adds a quantitative dimension to the argument in support of active learning. If CS students tend to be active and 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 An Active Approach to Computer Science 2

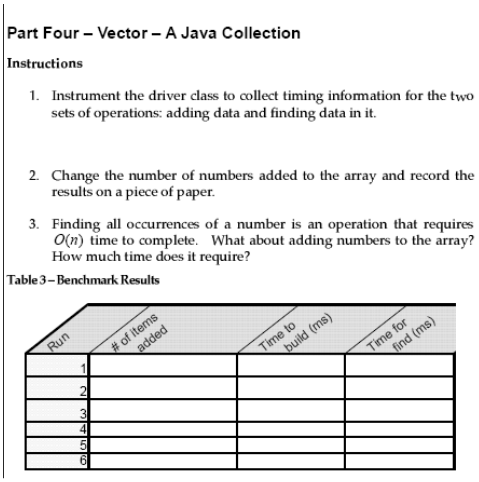
Computer Science 2 (CS2) is four credits and is the second course in our core program. The typical student is a freshman or sophomore CS major. The course objectives include using advanced data structures, abstract data types, and object oriented design. The course is taught using Java and the Eclipse IDE and an objects first approach. In objects first, object design, inheritance, abstraction, and design patterns are integrated into to the traditional data structure and ADT material.

The active techniques used in this course include many of the same techniques that appear in the literature such as peer review, paired programming, and small group problem solving [3]. We created a student lab manual to promote an active environment, to fill gaps in the material in the text book and to deliver material that appealed to a wider range of learning styles. The novel aspects of the lab manual include task based reference sections, interactive lecture/labs, and selected source code examples.

#### 2.0.1 Organization of the Lab Manual

The primary goal of the manual was to organize course material into a single printed source that especially appealed to the most common learning styles (visual and active) [6]. The lab manual was divided into four primary sections *Technology Manual*, *Reference Section*, *Guided Labs* and *Sample Code*. The final printed manual was 270 pages. The manual was sold by the campus book store as a required text to supplement the text book. The cost of the manual was nominal and recovered the printing and binding fees.

The Technology Manual contained instructions for using the various development environments, computer systems, and off-campus resources. For example, students used Sun workstations in the course, and the technology section included instructions using Sun's windowing manager (CDE). The technology section also included detailed step by step instructions for using Eclipse. The instructions were organized by the tasks the students would encounter. Examples



**Part Four – Vector – A Java Collection**

**Instructions**

1. Instrument the driver class to collect timing information for the two sets of operations: adding data and finding data in it.
2. Change the number of numbers added to the array and record the results on a piece of paper.
3. Finding all occurrences of a number is an operation that requires $O(n)$ time to complete. What about adding numbers to the array? How much time does it require?

**Table 3 – Benchmark Results**

| Run | # of Items added | Time to build (ms) | Time for find (ms) |
|---|---|---|---|
| 1 | | | |
| 2 | | | |
| 3 | | | |
| 4 | | | |
| 5 | | | |
| 6 | | | |

Figure 1: Students make empirical observations throughout the lab.

include: *Import External "jar" File*, *Recover Previous Versions of Files*, *Automating Getter and Setter Method Creation*.

The lab contained eleven lab exercises and three projects. Each lab corresponded to a chapter of the text book, and augmented the material in the text. The labs followed a progression such that the early labs were very specific and included detailed step-by-step instructions while later labs were very general, and involved more independent problem solving skills. The labs were designed to instruct and challenge student perceptions of the material and guide student's problem solving and programming skills.

The labs are a novel departure from the traditional in-class lab. The lab manual contains pre-printed copies of source code examples, key formulae, and space to record observations. Lectures and labs were frequently interleaved, such the lectures either prepared students to work on the lab, or review results of a completed section of the lab. During the lab portion of the class students worked with peers to solve the given problems. Students frequently discussed challenging parts of the material with each other; and often used the time to ask questions they were afraid to ask in front of the whole class.

One of the labs explored the relationship between the theoretical runtime of algorithms and the empirical runtime. The first part of the lab instructed students to devise methods to observe and compare the runtime behavior of certain data structures to the theoretical behavior. The second part asked students to code and execute their tests and record their empirical results using their own coded data structures and selected members of the Java Collections API (see figure 1). To help explain the apparent contradictions, students were directed to the source code appendix of their lab manual which included the relevant Sun data structure source code. They were able to see how professional programmers implemented the same data structures they did. This type of interaction helped students remain active while challenging them to explain empirical results that were not supported by the theory and help build strong mental models of the material.

### 2.0.2 Results

The results compare two Computer Science 2 courses, one in Fall 2003 and the other in Spring 2004. Both courses had the same instructor, textbook, and classroom resources, and had similar enrollments and student backgrounds. One difference is that four students in the Spring were repeating a failure from the previous Fall. The conventional teaching methods used in the Fall such as lectures, labs, and homework assignments that worked well for traditional structured

| Grade | Fall 2003 | Spring 2004 |
|---|---|---|
| A | 23% | 26% |
| B | 9% | 19% |
| C | 14% | 26% |
| D | 23% | 7% |
| F | 20% | 11% |
| W | 11% | 11% |

Table 1: CS2 Final Grades Comparison

programming were not well suited for the objects first approach. Students in the course were frequently frustrated with the complexity and scope of the material, and the general tone of the course was largely negative.

An end of term post-mortem analysis of the results of the Fall 2003 course showed that there were problems. The student's final grades (Table 1) show that nearly one-half of the students either dropped out (W), failed (F), or received a poor grade (D). Student feedback, collected through a standard University-wide 'student satisfaction' survey, included statements such as "projects were too large and complex" or "spend less time covering logic and more time programming."

A similar end of term post-mortem of the Spring 2004 course which relied on the active techniques shows an improvement in the final grades. Student feedback is the clearest endorsement. One student responded that the most positive aspect of the course is that the "labs get really involved and really test your skills" and another wrote "this is more hands on and challenging than any other CS course I've taken previously, which really helps [me] further understand the subject matter."

The active learning environment in Computer Science 2 helped transform the class into an interesting, challenging, and interactive environment. The effort that was put into assembling the manual was well worth the end results. This technique was effective, and well liked by the students.

# 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. Techniques such as frequent in-class problem solving, lab sheets and discussions were used to create an active environment to appeal to a broad range of learning styles. The active learning approach helped students move up from the lower levels of Bloom's taxonomy (simple knowledge and comprehension) into the highest levels (analysis, synthesis, and evaluation). Students attained a greater level of understanding of the material because they had the opportunity to interact with it demonstrated through test scores and student feedback. 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 is strong evidence in support of the efficacy of these methods. Student outcomes assessment shows that there was an improvement as a result of these techniques. 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. Standard student evaluation instruments asked students to identify the most positive aspects of the course. Over 60% of responses listed the in-class labs and class discussions as the most positive aspects of the course.

Active learning techniques can improve the dynamics of a class. The techniques appeal to a variety of preferred learning styles and remove impediments to cognitive processing. Most importantly, we are a publicly funded liberal arts university, and each of these techniques was developed and employed with little cost beyond the class preparation time of the instructor.

Future work will include development of new techniques and application to a wider variety of courses, including Operating Systems and Computer Organization. Active learning creates an environment that helps students share the instructor's enthusiasm for the material and builds confidence in the students.

# References


[1] Owen Astrachan. Concrete teaching: hooks and props as instructional technology. In *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*, pages 21–24. ACM Press, 1998.

[2] R. M. Felder and R. Brent. Learning by doing. *Chem. Engr. Education*, 37(4):282–283, Fall 2003.

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

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

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

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

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