Problem-Based and Case-Based Methods in Computer Science

D. E. Stevenson

Director, Institute for Modeling and Simulation Applications

Department of Computer Science

Clemson University

PO Box 340974

Clemson, SC 29634-0974

(864) 656-5880 / (864) 656-0145

steve@cs.clemson.edu

Jennifer Parham

Graduate Student

Department of Computer Science

Clemson University

PO Box 340974

Clemson, SC 29634-0974

(864) 656-5880 / (864) 656-0145

jparham@cs.clemson.edu

Dr. Stevenson received his A. B. in Mathematics from Eastern Michigan University in 1965, M.S. from Rutgers in computer science in 1975 and his Ph. D. in Mathematical Sciences from Clemson in 1983. He is a combat veteran of Viet Nam and worked from 1969-1980 at Bell Telephone Laboratories. Dr. Stevenson is a leading educator in modeling and simulation as a board member of the Shodor Education Foundation and a principal PI on the National Computational Science Institute (NCSI). He has been at Clemson since 1980 where he conducts research in modeling, simulation, and programming languages.

Jennifer R. Parham – PhD Candidate in Computer Science at Clemson University. Ms. Parham received her B.S. in Computer Science from Appalachian State University in 1999 and her M.S. in Computer Science from the University of Montana in 2003. As an adjunct lecturer at the Colorado Mountain College, Ms. Parham designed a curriculum for computer classes as part of a grant sponsored by the Colorado Women's Foundation. From 2001 to 2003, she led outreach and education initiatives as part of her work on NSF and Department of Energy EPSCoR grants at the University of Montana. Ms. Parham began work on her PhD at Clemson in 2004, and successfully completed her portfolio/comprehensives in January 2005. She is currently engaged in dissertation research, which emphasizes schema theory, the idea of organized knowledge as a network of abstract mental structures representing understanding, as it applies to the field of computer science.

Problem-Based and Case-Based Methods in Computer Science

Abstract

We discuss how problem-based and case-based learning (PBL/CM) principles are used to design and conduct an undergraduate computer science capstone project course. The students are placed in a simulated professional environment with carefully planned exercise that lead them to a design of a thousand line program. We use three different assessment tools to give insight about students' learning styles and higher order thinking skills, as well as to evaluate their learning under a PBL/CM environment. In addition, we use these assessment tools to find correlations between student's final grade and the score they received on a measurement tool that is not used in the calculation of their final grade for the course.

Observations about Students

Computer science poses a challenge to inquiry-based learning pedagogies because, unlike the other sciences, computer science takes problems from a wide spectrum of unrelated disciplines. By its very nature, computer science is about problem-solving a wide-spectrum of non-computer science problems by a very specific means: algorithms written in any one of a myriad of programming languages. In professional practice, programs can be millions of lines of code and take three to five years to produce. The class we have chosen to use in our work is the development of a compiler, one of the types of problems

Partial support for this work was provided by the National Science Foundation's National Science, Technology, Engineering, and Mathematics Education Digital Library program under grant **0435187**.

that takes multiple years to produce in practice.

Thus computer science is a problem-solving science. The conduct of professional computer scientists is typified by system development skills and requires self-assessment. Academic computer science, on the other hand, teaches personal skills early in the curriculum: programming, tools such as editors, and fundamental data structures. Typically, students are not required to conduct meta-cognition or self-assessment exercises. System development skills are needed during analysis and synthesis phases, usually such work is conducted in large groups. Typically, computer science curricula do not engage students in such large system development situations.

Personal Skills

Anecdotal evidence from Clemson student interviews indicates that many students reach the senior year without having gained significant problem-solving and design skills. This is often accompanied by poor study skills and improper time management. Students can remain passive and obtain good, if not excellent, grades. Examinations in various courses show that the students have a limited technical vocabulary. By the senior year, students have not gained significant experience in evaluating design alternatives when it comes to algorithm and system development. In short, students live at the lower levels of Bloom's taxonomy.

In an effort to quantify these findings, we have been using two instruments: The **Test of Logical Thinking (TOLT)** and Felder's **Individual Learning Styles** inventory. A learning styles test is administered for the benefit of the students. We no longer collect

information for statistical analysis having learned several years ago that the myth of computer science learning style is just that: a myth. Our examination showed that the learning styles are randomly distributed about the four axes, which are seeing and hearing, reflecting and acting, reasoning logically and intuitively, analyzing and visualizing, of the Felder test (see Figure 1). Anecdotally, we have noticed of late that the scores are more tightly clustered about the origin. We intend to explore this in the near future. The students do gain some insight from taking and analyzing such learning styles tests, however.

"Figure 1 here"

We have only recently begun using Tobin and Capie's test of logical thinking, a test often used by science faculty. The interest here is to find an instrument that measures a student's ability to deal with abstraction.

Large System Development Skills

Our students do not have an opportunity to work on large system development projects during their undergraduate experience at Clemson. For this purpose, a **large system** is one that has multiple program units and requires a significant programming effort, say one thousand lines at a minimum. A large system requires students to engage in project management. The only students to have such skills are non-traditional students with significant workplace experience.

One of the major issues in large system development is time management. The time required to develop a software system is based on estimating how many lines of code

(LOC) are required. Estimating LOC is a notoriously difficult task but extremely important. Industry-wide studies of productivity consistently show that programmers produce 2.5 debugged lines of code per hour. Hence, a system of one thousand lines would take approximately 27 hours per week in a fifteen week semester or a total of 400 hours. This 2.5 lines per hour is also reported by our students, with the very best only obtaining approximately 3 lines per hour.

Using self-reported time data, we find that the students have very little grasp of time management. We have introduced a limited version of Watts Humphrey's *Personal Software Process (PSP)* approach; the time management and planning portions have been abstracted. The PSP forms for time management have been augmented by a form to report on the entire week. As the anecdotal evidence suggests, the results of this exercise is an eye opener for the students.

Foci for CS Courses

Based on the situation reported above, Stevenson has developed several problem-based / case-based learning (PBL/CM) courses of the past five years. The principles involved in the development of PBL/CM courses are outlined in this section. We illustrate these principles in the context of a senior-level computer science class in programming languages required of all BS in CS students. The goal of this course is to develop an understanding of the nature of programming language features and of the compilation process.

Constructivity

Interestingly, the term *constructivity* has meaning in the context of computer science. Constructivism is the philosophical foundation of computer science, inheriting its meaning from mathematics: all problems can be solved with finite algorithms, finite objects, and certain fundamental processes.

Constructivity in the educational setting means the development of understanding of the subject by the learner by which the learner "constructs" her or his own knowledge structures, in our case understanding of formal and natural language and the compilation process. Epistemologically, this is **implicit** knowledge held internal to the individual, although occidental philosophy does not accept implicit knowledge as knowledge. A future research question is the role of implicit knowledge in assessment. Furthermore, knowledge in science is both consensual and communal; hence learners are not free to develop unwarranted, unjustified "knowledge." But there is more to knowledge that explicit, propositional knowledge. **Useful knowledge** is knowledge that one can bring to bear on a problem; in computer science, we can break useful knowledge into three parts:

1. **Propositional knowledge** is generally regarded as what education produces in the individual. For our purposes, we identify propositional knowledge as being structured by **concept maps.**
2. **Implicit knowledge**. Implicit knowledge is knowledge brought by the problem solver to the problem that is internal to the problem solver. Implicit knowledge is "experience" and it does play a role in development.

3. **Algorithmic knowledge.** Algorithmic knowledge is our term for implicit knowledge that is explicitly oriented toward computational issues. Specifically, we conjecture that algorithmic knowledge is held as **schemata** [Marshall].
    a. Programming is pattern recognition that trigger schemata.
    b. Understanding of constraints and criteria
    c. Planning.
    d. Implementation.

    In the educational literature, algorithmic knowledge seems to be associated with arithmetic operations. We mean something vastly different here.

Problem Solving Skills

Professionals in the computational sciences are required to innovatively solve problems in science and industry using very specific concepts, methods, and equipment. To do so, students must acquire expertise. “Expertise is an ability to make sound judgments as to what is problematic about a situation, to identify the most important problems, and to know how to go about solving …them” as related by Margetson [Margetson, p. 44]. While we understand much about the psychological aspects of problem-solving, acquiring these skills comes only by practice. Therefore, our concept of course development is that the conduct must provide opportunities leading to attainment of expertise.

Stevenson PBL/CM

With the above as prologue, the question is, “How does one, in the university setting, provide opportunities to obtain expertise?” We conclude from the research literature that

we need all three approaches: lecture, problem-based learning, and case-based learning. Lecturing is required during the course of a large project when communication of results or transfer of information dominate. PBL is an efficient way to learn when there is a fuzzy problem that must be completed outside class: A one thousand line computer system spread over fifteen weeks certainly qualifies. In fact, the project must be broken into eight to ten fuzzy problems that are called **milestones** in the project management literature. Case-based methods are required when the decision-making process is judgment based and not simply a formula to be verified. This interpretation of case-based methods may be slightly at odds with the literature, but it fits because each milestone must be further broken down into self-contained issues that require design judgment to proceed. For many reasons, we have chosen a "mixed metaphor" model. It seemed more appropriate to return to psychological principles.

1. Learners construct models to explain their worlds. But these constructed modules must first constitute knowledge (justified true belief) and the learner must explain the model to others. A simple quote attributed to Einstein tells the story, "You don't understand anything until you can explain it to your grandmother."
2. Learning must be intentional. When learners articulate what they have learned and reflect on the process and decisions of the processes they understand more and are better able to use the knowledge gained. This indicates that there must be many opportunities for meta-cognitive exercises.

3. The educational setting must be authentic, complex and contextualized. Research shows (Doub, 1991) that learning situated in some meaningful real-world task are not only better understood but more consistently transferred to new situations.
4. In computer science, the professional setting is cooperative, collaborative, and conversational. Collaboration most often requires conversations among participants who must socially negotiate a common understanding of the task and the methods they will use to accomplish it. In order to understand, students must search for meaning.

General Guidelines for Course Development

We have developed five guidelines to developing PBL/CM courses.

1. PBL/CM pedagogy. Choose a suitable project that can be broken into milestones, a milestone representing a *problem* in the PBL sense. Each milestone is broken into specific project objectives chosen within learning objectives. Cases are developed to lead learner investigation. Lecture is a last resort, and then it is best to require the students to provide some if not all the material. For "skills" courses such as data structures, choose a problem that is historically meaningful in the development of the data structure but choose it in a setting in which students must "dig it out."
2. Query-based conduct. The purpose of the classroom sessions is to develop understanding through the Socratic method. This approach requires a strong personality: lower level students hate, for many reasons, the Socratic approach.

Good students love it because it allows them to strike out on their own as well as a more open relationship with the instructor.

3. Evidence-Based Reasoning. In the way of this course, critical thinking is synonymous with evidence-based reasoning. A design should be reasoned and the implementation tested. Students in this course are graded on how well they test their milestone implementations.
4. Performance-Based assessment. Every student does every exercise. Groups are for discussion and critical thinking – not for doing a group solution to homework. There is much to be gained for completing the entire project even if the learner only does the coding. I (Stevenson) strive to make the projects such that the code-only student cannot pass the course.
5. Continuous Feedback. PBL and CM can be frightening to the students. A way to keep this under control is to be sure you have a good, continuous feedback mechanism, such as a one minute paper with three questions to be answered: (1) What was right today? (2) What was wrong today? And (3) What is your highest priority issue?

Course Development

In order to make the above desiderata come alive, it is necessary to consider the salient features of both the problem-based learning and case-based learning doctrines. There are three aspects: PBL, CM, and conduct.

The PBL Aspect of the Solution

PBL requires large amounts of resources in terms of teaching assistants. Those resources simply are not available at Clemson, whether they are available at any public U. S. university in these times is doubtful. Thus we must incorporate documented pluses of PBL without the traditional, labor-intensive setting.

PBL encourages open-minded, reflective, critical, and active learning by having the learner focus on a problem that must be solved. This PBL aspect meshes well with the CM approach as they each refer to a different aspect of a problem. The recommended method for choosing problems is to consider historical issues first: learners do not have a great understanding of the how disciplines have arrived at their current state of development.

1. Students and instructors must share a desire to develop knowledge, understanding, feeling, interests. **How:** instructors can allow, within limits, the students to set the agenda. The instructor must retain responsibility for the overall project, but students many opportunities to affect the detailed design.
2. PBL emphasizes the nature of knowledge as consensual and communal. **How:** Students must be required to use many outside sources. The course described here does not have a required text book but rather any of a number of good texts in the area may be used. A saying has evolved: “Google is your friend” because detailed questions in computer science educational projects are explored on the Internet;

fortunately, these discussions come to different conclusion – some even erroneous. The class must come to a consensus integrating these experiences.

3. Students must gain propositional knowledge. **How:** This requires careful selection of reading and subproblems. There are two aspects to this: students must gain propositional knowledge of the subject (programming languages) but they must also determine propositional knowledge of the project itself. The pedagogical question here is to recognize when the learners must be left to their own devices and when to step in.
4. PBL should help the learner gain the ability to analyze, synthesize, and criticize constructive solutions. **How:** This is where the case method comes in.
5. Both PBL and CM put propositional knowledge to best use in constructing a solution. **How:** Dewey's problem-solving paradigm is emphasized during the problem-solving process.
6. A principal professional value for computer scientists must be testing solutions for applicability and correctness. **How:** Computer science testing methodologies seem not to be emphasized in earlier courses and the students often struggle due to the formalisms and the lack of experience deriving tests. They are required here.
7. PBL Encourages Kanbrain. **Kanbrain** - the educational equivalent to *just in time (JIT) manufacturing* - introduces a topic at the correct time for maximum impact. (Perelman). Kanbrain is opposite of *just in case (JIC) learning*; much of lower division is taught "just in case you need it later."

The Case Aspect of the Solution

While problem-based learning dictates that students have a problem to solve, PBL does not say much about **how** to advance to a solution. **Case-based methods** address precisely this issue. Case-based approaches are about professional judgment, emphasizing analysis, synthesis, and decision-making. In the combined PBL/CM approach, the PBL side sets the problem but the cases (subprojects) are specifically laid out to develop these higher order critical thinking skills.

In our interpretation, case methods have the property of presenting realistic situations that require judgment calls about the design element; the case is well-circumscribed by this point. To be useful, the problems must have multiple possible solutions that require detailed analysis and solution synthesis. **How:** The instructor must encourage discussion on the choice of program elements, representation, and algorithms.

Case-based methods encourage deep analysis. This analysis along with the construction of a program and its testing help students develop understanding the problem's solution. **How:** Put the students are in charge.

Setting up the PBL/CM Tasks

The above discussion is just philosophy unless there are sound guidance rules for developing the individual exercises and subprojects. Our rules are

1. The problems and cases must be embedded in realistic, complex problems. **How:** A compiler is a very large and complex system. While not something an average organization would develop, it represents a pivotal concept in computer science. Compilers also make use of several important formalisms.
2. The problems and cases cannot be rigidly defined just to exercise a single recently learned skill or recently discussed fact. **How:** The project is built on milestones with each milestone constituting a problem. The milestones build on one another in complex ways. Each milestone is not just one problem but must be decomposed into sub-problems that can be analyzed and programmed.
3. The problems and cases are incompletely specified. **How:** The specification of the project (the requirements) is in a memo written to the students, with inadvertent errors as well as "malice of forethought" errors. The memo is "corrected" throughout the semester, but each new class gets the original, uncorrected memo at the beginning of the project. Furthermore, many very complex issues arise as the learners attend to details of the project.
4. There must not be a clear stopping point. **How:** This interjects considerable uncertainty because the learners do not know in advance how long to analyze or design. This is the common situation in professional practice where the project management dates dictate how complete analysis and synthesis are.
5. Real problems in computer science require interacting disciplines and skills. **How:** Students must use programming skills, data structure knowledge, and understanding of hardware. Programming languages adds an interesting interaction between philosophy (linguistics) and computer science.

6. Project oriented assignments emphasize a broad range of problems:
    a. Rule-using. Morphology and syntax are fundamental issues that are inherently rule-based as is type checking.
    b. Decision Making. Deciding among various options for representations and operations.
    c. Troubleshooting. Debugging. Re-visiting previous milestones, re-assessing previous decisions.
    d. Diagnosis-Solution. Debugging, Re-design.
    e. Case/system analysis. Each milestone is more than one problem.
    f. Design.
    g. Dilemmas. Contradictory statements. Requirements that are mutually exclusive.

Actual Course Conduct

The actual course develops a minimal compiler that requires the learners to master a new language (Forth) that represents the hardware interface; Forth is different from any other language the learners have encountered. The project requires six to eight milestones (problems), each milestone is approximately two weeks duration. The remainder of the semester is an exploration of the issues attendant to programming language design. The learners must understand that programming languages are languages first and programming directives second.

The learners are not prepared to develop a full semester project from scratch; the milestones are given to them. The milestones are further broken into cases, the answers to which constitute the design of the milestone.

The day-to-day conduct of the course follows a Socratic approach with a twist: the learners get to ask the questions first. We call this approach “query-based conduct.” This approach is based on the concept that the purpose of classroom time is for planning, discussion, and feedback.

The students are placed into two groups: a permanent one and one or more temporary ones. The permanent group is assigned the second day of class. The learners are randomly assigned, based on grade point average, but an attempt to make the groups have the same average grade point average. Students must actively participate individually and in groups. We do observe minority status and try to avoid isolated minority students; this has not been a problem.

Grading

The grading scheme is based on the following breakdown:

1. There are no tests given except a university mandated final. Students who have achieved a “A” average on their project and completed the requisite number of milestones are exempt.
2. Student milestones are graded on three axes:

a. Their code is handed in electronically. The grader compiles the code and runs it. If the code fails to compile or run, the students get zero points.
b. The student program is run against the student's own test files. If the code runs successfully against their own tests, the student receives 50 points.
c. The students' test data is compared to the instructor's and grader's tests. If the student provides at least that level of sophisticated tests, the student receives a total of 75 points.
d. Finally, the students turn in a meta-cognitive exercise on two questions: (1) How did you apply the design criteria to develop your program and (2) What did you learn.

3. The on-time portion of the milestone is graded separately. Students who have a net "early" days at the end of the semester can receive up to 10 points on their final grade. Students who have a net "late" days are penalized up to 10 points on their final grade. The total number of days that can be applied to this criteria are known before the project starts.
4. Milestones must be completed in order. No milestone can be handed in for credit after the next milestone has been handed in. However, student milestones must be handed in; this is to curtail cheating.
5. Students are expected to be in class every class period. Class attendance and homework constitute 30% of the final grade. Milestones (project) account for 60%, which includes the time calculation. The University requires a final exam unless the student is exempted by having an A average – a carrot, for sure.

Assessment

We gave several assessment questions. We wanted to determine if our students were formal thinkers (as defined by Piaget), if they could answer germane questions from the Computer Science Graduate Record Examination, how these tests correlate to the student's grade in the class, and how the PBL learning environment affects the students' learning. Finally, we have a large number of anecdotal comments on exit interviews. Until recently, the exit interviews and end-of-course questionnaires varied widely. This past semester (Spring 2005) was the first semester that all exit interviews were either neutral or positive – with the vast majority positive (see Table 1). We are unable to compare our methods with other sections because Stevenson is the only professor teaching this course.

"Table 1 here"

Statistics from Course Assessment

Currently, we have approximately 25% of our course drop, fail, or withdraw; on the other hand, approximately 30% receive As. Computer science in the bachelor of science curriculum student's are required to complete this course before graduating with a Computer Science degree, but this course becomes a major hurdle for many students because of the problem-based and case-based teaching style of the instructor. This has prompted the quest for alternative assessment tools to determine (1) what is linked to the students' grades, (2) whether the alternative teaching style increases the student's knowledge, and (3) what is the state of their current cognitive ability. The data was collected from Spring 2004 through the Spring 2005 semester, and all of the statistics

were run using the SAS 9.1 Statistical Package. Our mean values were obtained using proc means, Pearson correlations were obtained using proc corr, regression models with p-values were generated using proc reg, and the pretest and posttest comparisons were obtained through the proc ttest SAS commands.

The first assessment tool used was the Test of Logical Thinking (TOLT), which was developed by Tobin and Capie from the University of Georgia. This test is a well-known test in the physical sciences, such as chemistry and physics, and it was based on Piaget and Inhelder's work done in 1958. TOLT measures students' ability to reason through formal thinking skills. By such tests we can determine if a student has the higher order thinking skills associated with the physical sciences and the overall level of thinking; i.e. concrete, low/high transitional, or formal. Following is an example of a TOLT question:

Are fat fish more likely to have broad stripes than thin fish?

1. yes, 2. no.

Reason:

1. Some fat fish have broad stripes and some have narrow stripes
2. 3/7 fat fish have broad stripes
3. 12/28 are broad striped and 16/28 are narrow striped
4. 3/7 fat fish have broad stripes and 9/21 thin fish have broad stripes
5. some fish with broad stripes are thin and some are fat

A picture is supplied with every question that is used to answer the question. A question is marked correct only if the student supplies the right answer and reason. Piaget and

Inhelder determined that a student's natural cognitive progression changes through major life changes, and a student should be at a concrete thinking level around age 7-10, the transitional period between 10-15, and formal level by the time you are 15 or older. We tested 66 students in the spring of 2004 and the spring of 2005 under the IRB#40206. We were interested in finding the overall level of logical thinking among our 400-level students, the level of thinking within each higher order thinking skill and question, how the TOLT test score, skills, and questions correlate to the student's grade, and lastly, how the TOLT test correlates to the Clemson Math Placement Test (CMPT).

We are pleased that our 400-level students' mean score indicates that the students are formal thinkers with a mean score of 81.2%. The most missed skill is Correlation reasoning, which is the determination of correlation among variables, with a mean score of 73.5%. In addition, Question #8, which is provided as an example in the above paragraph, uses correlation reasoning was the most guessed question, where the student answered the question correctly with the wrong reason. The least missed skill is Probability reasoning with a mean score of 90%. The most missed question is Question #2, which is a Proportional question, with a mean score of 70%, and the least missed question is Question #5, which is a Probabilistic question, with a mean score of 92%. The overall TOLT score is not useful in predicting the student's grade. However, Correlation reasoning is helpful in predicting a student's grade with 91% confidence, i.e. ρ-value=.09, and more specifically, Question #8 has the highest correlation with the student's grade with a ρ-value=.01. However, Probabilistic reasoning and Question #6, which is Probabilistic reasoning, show an inverse correlation with the student's grade, i.e.

the lower the score, the better the grade, which is shown by a negative Pearson Correlation Coefficient. Even though some of this information is interesting, the mean score indicates that the students are at the formal level of thinking. Therefore, we decided to test the correlation between the TOLT test and the Clemson Math Placement Test (CMPT).

The Clemson Math Placement Test is used to gauge the incoming student's preparedness for calculus. The test has a long history of use at Clemson. The test has two parts, an algebra and a pre-calculus section. The CMPT algebra score, pre-calculus score, and the combined score of both show a strong correlation with the student's grade in our class with a Pearson Correlation of approximately .55, ρ-value=.01, and the R-Square, which is the amount of variability reduced by the regression model, is approximately .30. However, neither the CMPT algebra score nor the pre-calculus score is helpful in predicting the TOLT score, and CMPT and TOLT have a very low Pearson Correlation of .25 with a ρ-value=.25.

Lastly, we decided to test the students' ability to attempt and/or answer correctly Computer Science Subject GRE questions before and after taking the class. We selected 14 questions from the GRE question bank, and we assigned a 1 for no answer, 2 for an attempt, and 3 for the correct answer. We tested 39 students from the fall of 2004 and spring 2005. This gives indication of the student's computer science knowledge as deemed important by the National Assessment Board at the senior level and the affects the course has on the students' knowledge. The pretest mean score was 71% and the

posttest mean score was 82%. Therefore, there was an increase among the mean GRE score with a ρ-value=<.00005. Every question's mean score increased except #s 1, 4, 9, and 12. The test is available on-line from several different venues; our copy came from Professor Carl Rotter's website [Rotter]. Neither the pretest nor posttest are helpful in predicting the student's grade, but the posttest with a ρ-value=.37 is more helpful than the pretest with ρ-value=.51. There are a few questions that show correlations with the student's grade, and these include pretest #5, 6, 8, and 10 and posttest #5, 7, 9, and 13. All ρ-values are less than or equal to .05 to be 95% confident, however, the Pearson Correlation is between .31 and .39 with an R-Square between .10 and .15.

In conclusion, we see that the Clemson senior computer science students are formal thinkers, and the data shows that TOLT is not a good assessment tool for computer science students at the senior level. However, it is interesting that Correlation reasoning is the weakest and most guessed skill, as well as linked to the student's grade in the course with 91% confidence. The statistics show that neither the overall TOLT nor GRE score is useful in predicting the student's grade. Whereas, the CMPT score (algebra and pre-calculus) seems more correlated to the final grade. It appears that the problem-based and case-based pedagogy increases the students overall Computer Science GRE score, which is optimistic for this teaching style.

Conclusions

The course has evolved over approximately ten years from a "traditional" lecture based class into one that uses all forms of delivery and emphasizes active learning in all aspects

of the course. This particular course has been a running experiment as Stevenson is the only person to teach it.

The road has not been easy and it has been difficult to gain student acceptance, especially since until a year ago I was the only professor using either PBL or CM techniques. Two changes seem to have drastically altered the student acceptance:

1. The incorporation of the personal software process portion, however small, showed many students, even exceptional ones, that their use of time was suboptimal.
2. Experiences of students during interviews for jobs have become "folklore" and students are concluding that these skills are marketable. Anecdotal evidence from student's returning from interviews is that the interviewers are very interested in the students' experiences in the class. Again anecdotally, the students indicate that the interviewers uniformly approve of the real-world nature of the course.

One problem with PBL/CM is the lack of textbooks. Stevenson has recently signed a contract with CRC to develop a textbook for the programming language course that should be available for the Fall, 2006 semester.

Student Interview Comments

Only problem was with 428, can't seem to hook up to get grades
Wish [I had] more design before I got to 428
Non-specific language courses were good preparation for 428

405 and 428 good, more figure out on your own, knowing how to teach yourself
I'll hate myself for saying this, but 428 has been a good class, tons of work but it lets you see what kind of programmer you are; I found out what I really could do
Hard time fitting with Socratic method, my learning style; don't like book for 428
428 is teaching me my potential about how much I can get done, would have liked to have had the experience earlier
Would be great to have something like we are doing in 428 earlier in program, maybe watered down some;
428 great prep for the real world; no cradle to grave projects except for 428
Yes – especially 428; learning how to learn; not sure about the other classes really doing that. Relate to real world - especially 428, owe a lot of that to Dr. Stevenson
If I hadn't had 428 I would say "no" – Stevenson is rough but wish I could have had a course like that earlier on
428 did that for me, nothing else has prepared me like that has
Yes, especially later classes, 428 especially
428 was very helpful
428 changed my view, while in the class I hated him, but at the end I realized what he was getting at, and it made sense;
Made mistake taking 422 and 428 same term
Enjoyed 428, it's hard but at the end of the day I've applied everything I've learned here
Two best classes have been 481 (Malloy) and Stevenson in 428. It really helped me a lot to pull the entire program together; really built my confidence
I think 350 should be required before taking 428

428 was awesome, really made me competitive in job market

Table 1

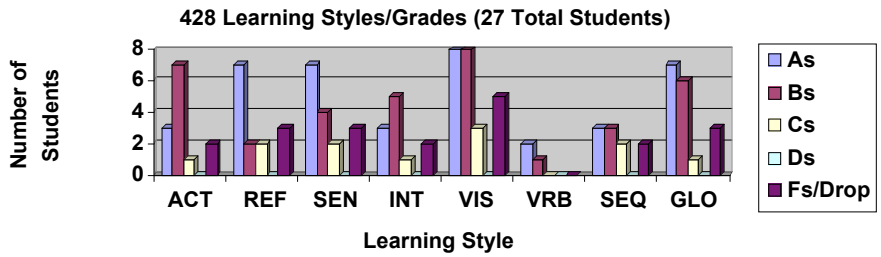

Figure 1 – These are the various learning styles in the classroom and their relationship w/ the students' grade.

The styles are as follows: Active, Reflective, Sensing, Intuitive, Visual, Verbal, Sequential, and Global

Bibliography

.Boud, D. J. (ed). *Problem-Based Learning in Education for the Professions.* Syndey: Higher Education Research and Development society of Australasia.

Boud, David and Feletti, Grahame. *The Challenge of Problem Based Learning*. New York: St. Martin's Press. 1991.

Bransford, John D., Brown, Ann L., and Cocking, Rodney R. *How People Learn: Brain, Mind, Experience, an dSchool.* National Academy of Sciences Press. 1999.

Jonassen, D. H.. Toward a design theory of problem solving. *Educational Technology:*

Research and Development. 2001.

.Fekete, Alan, Greening ,Tony, and Kingston, Jeffrey. Conveying Technical Content in a Curriculum Using Problem Based Learning. In ACSE98, Brisbane QLD Australia. 1998. 198-202.

Felder, Robert & Soloman, Barbara. Index of Learning Styles Questionnaire. NC State University. *June 29, 1999.* http://www.engr.ncsu.edu/learningstyles/ilsweb.html.

Inhelder, Barbel & Piaget, Jean. Parsons, Anne & Milgram, Stanley. (Trans.). (1958). *The Growth of Logical Thinking from Childhood to Adolescence.* USA: Basic Books, Inc.

Margetson, Don. Why is Problem-based Learning a Challenge. In Boud David and Feletti, Grahame. *The challenge of Problem-Based Learning.*

Nickles, T. Introductory essay: scientific discovery and the future of philosophy of science. In Scientific Discovery, Logic, and Rationality. *T. Nickles (ed). Dordrect: D. Reidel.*

Marshall, S. P. (1995). Schemas in problem solving. Cambridge: Cambridge University Press.

Rotter, Carl.

http://www.as.wvu.edu/coll03/phys/www/rotter/phys201/1_Habits_of_the_Mind/Test_of_Logic_Thinking.html

Tobin, K. & Capie, W. (1981). Development and validation of a group test of logical thinking. Educational and Psychological Measurement, 4l(2), 4l3-424.