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 for the National Computational Science Institute (NCSI) grant. He has been at Clemson since 1980 where he conducts research in modeling, simulation, and programming languages.

Jennifer R. Parham is a 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, emphasizing 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 (PBL) and case-based learning (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 exercises that lead them to a design of a thousand-line program. We use three different assessment tools to provide 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 students’ final grades and the score they received on a measurement tool that is not used in the calculation of their final grades for the course.

Observations About Students

Computer science poses a challenge to inquiry-based learning pedagogies because, unlike the other sciences, the field takes problems from a wide spectrum of unrelated disciplines. By its very nature, computer science is about problem-solving a problem using 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 subject we have chosen for our work is the development of a compiler, one of the types of problems that takes multiple years to

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.
produce in practice. Academic computer science often teaches individual skills but not
the skills professionals are expected to use in the workplace. Students are not often
required to conduct meta-cognition or self-assessment exercises, yet this is common in
professional practice. Our work reported here develops a more professional, integrated
experience.

**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 and time management skills. Students can remain
passive and obtain good, if not excellent, grades. Even by their senior years, students
have a limited technical vocabulary as evidenced by their examinations and have not
gained significant experience in evaluating design alternatives in relation 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)* [14]and Felder’s *Individual Learning Styles* [6] inventory,
administered for the benefit of the students. We no longer collect information for
statistical analysis from the individual learning styles inventory, because statistically we
found 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, (see Figure 1). Anecdotally, we have observed of late that the scores are
more tightly clustered about the origin, suggesting no preferred learning style. The
students do gain some personal 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 Tobin and Capie [13][14] instrument examines the students’ reasoning level according to Imhelder and Piaget [7]. The focus in Tobin and Capie 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 purposes of this paper, 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 students having such skills are often 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, a difficult task but extremely important task. 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 takes 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 2.9+ lines per hour.
Using self-reported time data, we found that the students have very little grasp of time management. As a result, we have introduced a limited version of Watts Humphrey’s \textit{Personal Software Process (PSP)} \cite{Humphrey1995-8} approach, focusing on the time management and planning portions. The PSP forms for time management have been augmented with a form for reporting an entire week. As anecdotal evidence suggests, the results of this exercise emphasizes for the students their lack of time management skills.

\section*{Foci for CS Courses}

Based on these situation, Stevenson has developed several problem-based / case-based learning (PBL/CM) courses over the past five years, these principles \cite{Stevenson2001-1,Stevenson2002-2,Stevenson2003-3,Stevenson2004-9,Stevenson2010-10} 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.

\section*{Constructivity}

The term \textit{constructivity} provides in the context of computer science provides the philosophical foundation inheriting its meaning from mathematics: all problems can be solved with finite algorithms, finite objects, and specified 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 the 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. 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 than 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 is the knowledge brought by the problem-solver to the problem; as such, it is the experience and it does play a role in development.

3. 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 [11].
   b. Constraints and criteria.
   c. Planning.
   d. Implementation.

In some educational literature, algorithmic knowledge is associated with arithmetic operations. We mean something quite different here.
**Problem-Solving Skills**

Professionals in the computational sciences are required to creatively solve problems in science and industry using very specific concepts, methods, and equipment. To do so, students must acquire expertise what Margetson [Margetson, p. 44] refers to as “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.” While we understand much about the psychological aspects of problem-solving, acquiring these skills comes from experience. Therefore, our concept of course development is that the conduct must provide opportunities leading to the attainment of expertise.

**Stevenson PBL/CM**

The question is how to provide opportunities to obtain expertise while in the university setting. We conclude from experience that we need 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 called milestones. Case-based methods are required when the decision-making process is judgment-based. This interpretation of case-based methods may slightly vary with the literature, but it fits because each milestone must be further broken down into self-contained issues that require design judgment to proceed. We have chosen the mixed metaphor model listed below as it seemed more appropriate to return to psychological principles.
1. Learners construct models to explain their worlds. However, these constructed models must first constitute knowledge (justified true belief) and the learner must explain them to others. A saying, attributed to Einstein, is operative, “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 reflect on the process, and reflect on the decisions made they understand more and are better able to use the knowledge gained. This process indicates that there must be many opportunities for meta-cognitive exercises.

3. The educational setting must be authentic, complex and contextualized. Research shows [1] that learning situated in some meaningful real-world task is not only better understood but also more consistently transferred to new situations.

4. In computer science, the professional setting is cooperative, collaborative, and conversational. Collaborations require 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 the following guidelines for developing PBL/CM courses.

1. PBL/CM pedagogy. A suitable project is one that can be broken into milestones, each representing a problem in the PBL sense. These milestones are then broken into specific project objectives chosen from the learning objectives, with cases being developed to lead learner investigation. Lecture is a last resort used only if students provide the questions or topics. For skills courses such as data structures,
a problem historically meaningful in the development of the subject is chosen in a setting in which students must “dig it out.”

2. Query-Based Conduct. The purpose of these classroom sessions is to develop understanding through the Socratic method. This approach requires a strong personality: while lower-level students dislike, for many reasons, the Socratic approach, good students often enjoy it because it allows them to strike out on their own as well as to develop a more open relationship with the instructor. Query-based conduct stimulates the class interactions and group mentoring.

3. Evidence-Based Reasoning. Critical thinking is included with evidence-based reasoning: a design should be reasoned and the implementation tested. Students are partially graded on how well they test their milestone implementations using their, not the instructors’, data.

4. Performance-Based Assessment. Every student does every exercise, groups being used for discussion and critical thinking, not for group solutions to milestone. There is much to be gained by completing the entire project even if the learner only does the coding. The instructor must be careful to make the projects such that the code-only student cannot pass the course.

5. Continuous Feedback. PBL and CM can be frightening to students. One way to prevent this is for the instructor to have a 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

To implement these desiderata in the classroom, it is necessary to consider the salient features of both the problem-based learning and the case-based learning doctrines.

*The PBL Aspect of the Solution*

PBL requires large number of teaching assistants. Since these resources are not available at Clemson, the documented pluses of PBL must be incorporated 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 each refers to a different aspect of a problem. The recommended method for choosing problems is to consider historical issues first: learners do not have a good understanding of the how disciplines have arrived at their current state of development.

Below are seven goals for achieving PBL aspects of the solution, followed in each case with “how”; i.e., a method for implementing each.

1. Students and instructors must share a desire to develop knowledge, understanding, feeling and interests. How: instructors can allow, within limits, the students to set the agenda. Even then the instructor must retain responsibility for the overall project, the students have many opportunities to affect the detailed design.

2. The nature of knowledge is consensual and communal. How: Students must be required to use many outside sources. The course described does not have a required
text book, allowing any of a number of texts. A saying has evolved: “Google is your friend” because detailed questions on computer science aspects are explored on the Internet; fortunately, these discussions come to different conclusions, some even erroneous. The class must come to a consensus integrating these experiences.

3. **Students must gain propositional knowledge. How:** This goal requires careful selection of reading and decomposing into sub-problems. There are two aspects to this process: students must gain propositional knowledge of the subject (programming languages), but they must also determine propositional knowledge of the project itself. The pedagogical issue becomes recognizing that the learners can 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:** These skills are developed by the case method.

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. **Testing solutions instills a principled professional value in the students. 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. Test development and implementation are required here.

7. **PBL encourages Kanbrain delivery. Kanbrain, the educational equivalent to just in time (JIT) manufacturing, introduces a topic at the correct time for maximum impact [12]. How:** 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 how to advance to a solution. Case-based methods address precisely this issue of professional judgment, emphasizing analysis, synthesis, and decision-making. In the combined PBL/CM approach, the PBL emphasis sets the problem but the cases (subprojects) are specifically laid out to develop these higher order critical thinking skills.

From the proposed perspective, case methods have the property of presenting realistic situations that require judgment calls about design elements. To be useful, these problems must have multiple possible solutions requiring 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, which along with the construction of a program and its testing, helps students develop an understanding of the problem’s solution. How: Put the students in charge.

**Setting up the PBL/CM Tasks**

This discussion is only philosophy unless there are sound guidelines for developing the individual exercises and subprojects. Ours include

1. The problems and cases must be embedded in realistic, complex problems. How: A compiler is a large, complex system. While not something an average organization
would develop, it represents a 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 constituting one problem. These milestones build on one another in complex ways. Each milestone although one problem must be decomposed into sub-problems that can be analyzed and programmed.

3. The problems and cases are incompletely specified. How: The specifications of the project (the requirements) are 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 class gets the original, uncorrected memo at the beginning of the project. Furthermore, many complex issues arise as the learners attend to details of the project.

4. There must not be a clear stopping point. How: The lack of a clear termination point interjects considerable uncertainty because the learners do not know in advance how long to analyze or design. Such a situation is 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 add 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 inherently rule-based. Type-checking is itself a formal reasoning, rule-based exercise.

b. Decision-Making. Deciding among various options for representations and operations based on time and space complexity.

c. Troubleshooting/Debugging. Re-visiting previous milestones, re-assessing previous decisions.

d. Diagnosis-Solution. This is another form of debugging that causes the redesign of elements.

e. Case/System Analysis. Each milestone includes sub-problems developed by case analysis. The solution of each sub-problem require judgments concerning time and space complexity of algorithms.

f. Design. The overall design of a large project requires both content knowledge in computer science and project management knowledge. A compiler is generally too large for undergraduates to design on their own; however, the students can participate in milestone design elements.

g. Dilemmas. Contradictory statements. Requirements that are mutually exclusive.

Class Sessions

The objective of the course is to develop a minimal compiler that requires the learners to master Forth, representing the hardware interface, different from any other the learners have encountered. The project requires six to eight milestones (problems), each is approximately two weeks in 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.

Since the learners are not prepared to develop a full-semester project from scratch, the milestones must be created by an experienced developer. These 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. This approach, called “query-based conduct,” is based on the concept that the purpose of classroom time is for planning, discussion, and feedback.

The students are placed into two types of groups: a permanent one and one or more temporary ones. For the permanent group, assigned the second day of class, the learners are randomly assigned, based on grade point ratio, but an attempt to make the groups have the same average grade point ratio. Students must actively participate individually and in groups. As minority status is observed, 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 an “A” average on their project and completed the requisite number of milestones are exempt.
2. Student milestones are graded on three axes:
   a. The 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 sophistication, 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 grades. Students who have a net “late” days are penalized up to 10 points on their final grades. 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’s due date has passed. However, student milestones must be handed in to help prevent cheating.

5. Students are expected to be in class every 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.
Assessment

Several assessment instruments were used to determine if the students are formal thinkers as defined by Piaget, if they can 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 environment affects the students’ learning. Finally, a large number of anecdotal comments during exit interviews (Appendix) were compiled. Until recently, the exit interviews and end-of-course questionnaires varied widely. Spring 2005 is the first semester that all exit interviews were either neutral or positive – with the vast majority positive (see Table 1). These methods could not be confirmed with other sections because Stevenson is the only professor teaching this course.

“Table 1 here”

Statistics from Course Assessment

Approximately 25% of our students who begin the course either drop, fail, or withdraw; on the other hand, approximately 30% receive As. Although Bachelor of Science students are required to complete this course before graduating, it becomes a major hurdle for many students because of the instructor’s problem-based teaching style. As a result, alternative assessment tools have been used 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 are from the Spring 2004 through the Spring 2005 semesters, all of the statistics were developed using the SAS 9.1 Statistical Package. The mean values are obtained using `proc means`, the Pearson correlations are obtained using `proc corr`, regression models
with $p$-values using \texttt{proc reg}, and the pretest and posttest comparisons through the \texttt{proc ttest} SAS commands.

The first assessment tool used is the Test of Logical Thinking (TOLT), developed by Tobin and Capie [14]. This test, based on Piaget and Inhelder, is often used in the physical sciences, measuring students' ability to reason through formal thinking skills used to determine if a student has the level of higher order thinking skills; i.e. concrete, low/high transitional, or formal. Below is an example of a TOLT question and we have omitted the accompanying diagram:

**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 diagram is supplied with every question that is used to answer the question. A question is marked correct only if the student supplies both the correct answer and the correct reason. Piaget and Inhelder determined that a student’s natural cognitive progression changes over major life changes, with a student being at a concrete thinking level between age 7-10, the transitional period between 10-15, and the formal level at 15
or older. In the springs of 2004 and 2005, 66 students were tested under the IRB#40206 to find the overall level of logical thinking among Clemson University senior computer science students, the level of thinking within each higher order thinking skill; how the TOLT test score, skills, and questions correlate to the student’s grade; and how the TOLT test correlates to the Clemson Math Placement Test (CMPT).

The mean score indicates that the students were formal thinkers with a mean scores of 81.2%. The most missed skill was correlation reasoning, which is the determination of correlation among variables, with a mean score of 73.5%. In addition, Question 8, the fat fish example seen above, which uses correlation reasoning, was the most guessed question, where the students 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 was Question 2, which is a proportional question, with a mean score of 70%, and the least missed question was Question 5, which is a probabilistic question, with a mean score of 92%. The overall TOLT score was not useful in predicting the student’s grade. However, correlation reasoning was helpful in predicting students grades at 91% confidence level; i.e. $p$-value=.09, and more specifically, Question 8 had the highest correlation with the student’s grade with a $p$-value=.01. However, probabilistic reasoning as exemplified in Question 6 showed an inverse correlation with the student’s grade. Even though some of this information is interesting, the mean score indicated that the students were at the formal level of thinking. Therefore, the correlation between the TOLT test and the Clemson Math Placement Test (CMPT) was tested.
The Clemson Math Placement Test is used to gauge incoming student’s preparedness for calculus. This test, which has a long history of use at Clemson, has two parts, an algebra and a pre-calculus section. The CMPT algebra score, pre-calculus score, and the combined showed a strong correlation with students grades in the class with a Pearson Correlation Coefficient of approximately .55, \( p\)-value=.01, and the \( R\)-Square, which is the amount of variability reduced by the regression model, was approximately .30. However, neither the CMPT algebra score nor the pre-calculus score was helpful in predicting the TOLT score, and had a very low Pearson Correlation of .25 with a \( p\)-value=.25.

Finally, the students’ ability to answer correctly Computer Science Subject GRE questions before and after taking the class, was tested using 14 questions from the GRE question bank, a 1 being assigned for no answer, 2 for an attempt, and 3 for the correct answer. From the Fall 2004 and Spring 2005, 39 students were tested to provide an indication of students computer science knowledge as deemed important by the National Assessment Board at the senior level and the effects of the course on this knowledge. The pretest mean score was 71\% and the posttest mean score was 82\%, showing a significant increase among the mean GRE scores with a \( p\)-value \leq 0.00005. The mean scores increased except for four questions (1, 4, 9, and 12). The test is available on-line from several different venues; ours came from Professor Carl Rotter’s website [13]. Neither the pretest nor posttest was helpful in predicting the students grades, but the posttest, with a \( p\)-value=.37, was more helpful than the pretest with \( p\)-value=.51. There were a few questions showing correlation with the students grades, including pretest
questions 5, 6, 8, and 10 and posttest questions 5, 7, 9, and 13. All \( p \)-values \( \leq 0.05 \) at a 95% confidence level, however, the Pearson Correlation was between .31 and .39 with an \( R \)-Square between .10 and .15.

In conclusion, the Clemson senior computer science students were found to be formal thinkers, and the data showed that TOLT is not a good assessment tool for computer science students at the senior level. However, correlation reasoning was the weakest and the most guessed skill, as well as linked to the students grades in the course with a 91% confidence level. The statistics showed that neither the TOLT nor the GRE score was useful in predicting the students grades, whereas, the CMPT score (algebra and pre-calculus) seemed more correlated to the final grade. It appears that the problem- and case-based pedagogy increased the student overall Computer Science GRE scores, suggesting this is an effective pedagogical tool.

**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. It has been difficult to gain student acceptance for this pedagogy, especially since until a 2005, Stevenson was the only professor in the Department using either PBL or CM techniques. Two changes seem to have significantly 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 students returning from interviews support the view 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 developed a textbook for this programming language course that should be available for Fall 2006.

**Student Interview Comments**

<table>
<thead>
<tr>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Only problem was with 428, can’t seem to hook up to get grades</td>
</tr>
<tr>
<td>Wish [I had] more design before I got to 428</td>
</tr>
<tr>
<td>Non-specific language courses were good preparation for 428</td>
</tr>
<tr>
<td>405 and 428 good, more figure out on your own, knowing how to teach yourself</td>
</tr>
<tr>
<td>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</td>
</tr>
<tr>
<td>Hard time fitting with Socratic method, my learning style; don’t like book for 428</td>
</tr>
<tr>
<td>428 is teaching me my potential about how much I can get done, would have liked to have had the experience earlier</td>
</tr>
<tr>
<td>Would be great to have something like we are doing in 428 earlier in program, maybe watered down some;</td>
</tr>
<tr>
<td>428 great prep for the real world; no cradle to grave projects except for 428</td>
</tr>
<tr>
<td>Yes – especially 428; learning how to learn; not sure about the other classes really doing</td>
</tr>
</tbody>
</table>
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>
<thead>
<tr>
<th>Number of Students</th>
<th>ACT</th>
<th>REF</th>
<th>SEN</th>
<th>INT</th>
<th>VIS</th>
<th>VRB</th>
<th>SEQ</th>
<th>GLO</th>
</tr>
</thead>
<tbody>
<tr>
<td>428 Learning Styles/Grades (27 Total Students)</td>
<td>8</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
</tbody>
</table>

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


http://www.engr.ncsu.edu/learningstyles/ilsweb.html.


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