**Urban-Lurain, M., Mishra, P., Koehler, M.J., & Banghart, R. (2001).** *Fluency with Information Technology (FITness): The Computer Science Perspective*. Paper presented at the 2001 Annual Meeting of the American Educational Research Association, Seattle, WA.

# Fluency with Information Technology (FITness): The Computer Science Perspective

Mark Urbain-Lorain
Punya Mishra
Matthew J. Koehler
Rick Banghart

Michigan State University

## Introduction

The report Being Fluent with Information Technology introduces the term Fluency with Information Technology (FIT). Persons who are FIT move beyond "training" to a deeper level of conceptual understanding that allows them to apply their knowledge of information technology to solving new problems in new domains and to learn to use new software as it becomes available. FITness requires three types of knowledge: (a) contemporary skills, the ability to use various computer applications; (b) foundational concepts, the basic principles and concepts of computing that form the basis of computer science; and (c) intellectual capabilities, the ability to apply information technology in particular situations and use this technology to solve new problems. The contemporary skills will change quickly over time, with the advances of computer software, while the underlying concepts are more stable. Intellectual capabilities are not restricted to a single course but should be developed throughout the undergraduate curriculum. Therefore, students should learn both skills with applications and computing concepts and principles so they can use computers to solve problems in a variety of disciplines after they leave the course. While this definition of FITness is one that many can support, the best curriculum to achieve these goals is not so clear. This paper summarizes how in the Department of Computer Science and Engineering at Michigan State University attempts to teach FITness and presents the results of analyses that suggest the approach is very successful.

### The Importance of Conceptual Understanding

It is only through conceptual understanding that individuals can transfer their knowledge about technology from the conditions in which they learned about technology to new situations, allowing them to adapt to rapidly changing information technology. Traditionally, computer science departments have taught computer programming in introductory computing courses. Programming was originally necessary to be FIT because all interactions with the computer required programming . However, interaction with information technology has now moved to higher levels of abstraction. For example, collecting and analyzing data used to require writing computer programs; today we use spreadsheets and databases to do these tasks with much greater ease and sophistication. Furthermore, learning theory calls into question the idea that conceptual understanding requires learning to program . So, if students do not program, how are they to learn and understand computing concepts? How can they learn more than a set of isolated skills that will be obsolete as soon as they leave the class?

For students to adapt to new computing systems and solve new problems, they must have a deeper understanding than is generally acquired by training with specific software packages; they need mental models or schemas of the computing systems they are using. Many introductory computer science (CS) courses for non-CS majors take a deductive approach that is adapted from the curriculum for CS majors. They present abstract computing concepts to students with the goals of having students a) learn the concepts, b) learn to identify a variety of disparate computing problems, c) link the problems to the underlying concepts and d) apply the concepts to the solution of the problems. This is a long chain of inference that has rarely been successful in courses for non-CS majors. Students come away with little understanding of the computing concepts and scant ability to solve new problems .

An alternative is to use an inductive approach. Rather than first trying to learn decontextualized concepts and then attempting to use those concepts to solve problems, course content can be organized to introduce students to computing concepts by having them solve a series of problems that epitomize classes of problems for which various computing concepts are the solutions . By structuring the course with a spiral curriculum which presents increasingly challenging problems to solve, students build from procedural skills towards conceptual understanding.

People learn concepts "as contextual entities (correlational structures), with common attributes that are the most typical, or average, members of a class" . As students grapple with apparently unrelated problems, the instruction must tie the problems together, showing how each is an example of particular concepts or principles. Students "triangulate" on these concepts and principles, refining their schemas as they solve successively more abstract problems, allowing their novice knowledge to evolve into expert knowledge . Figure 1 represents the process.

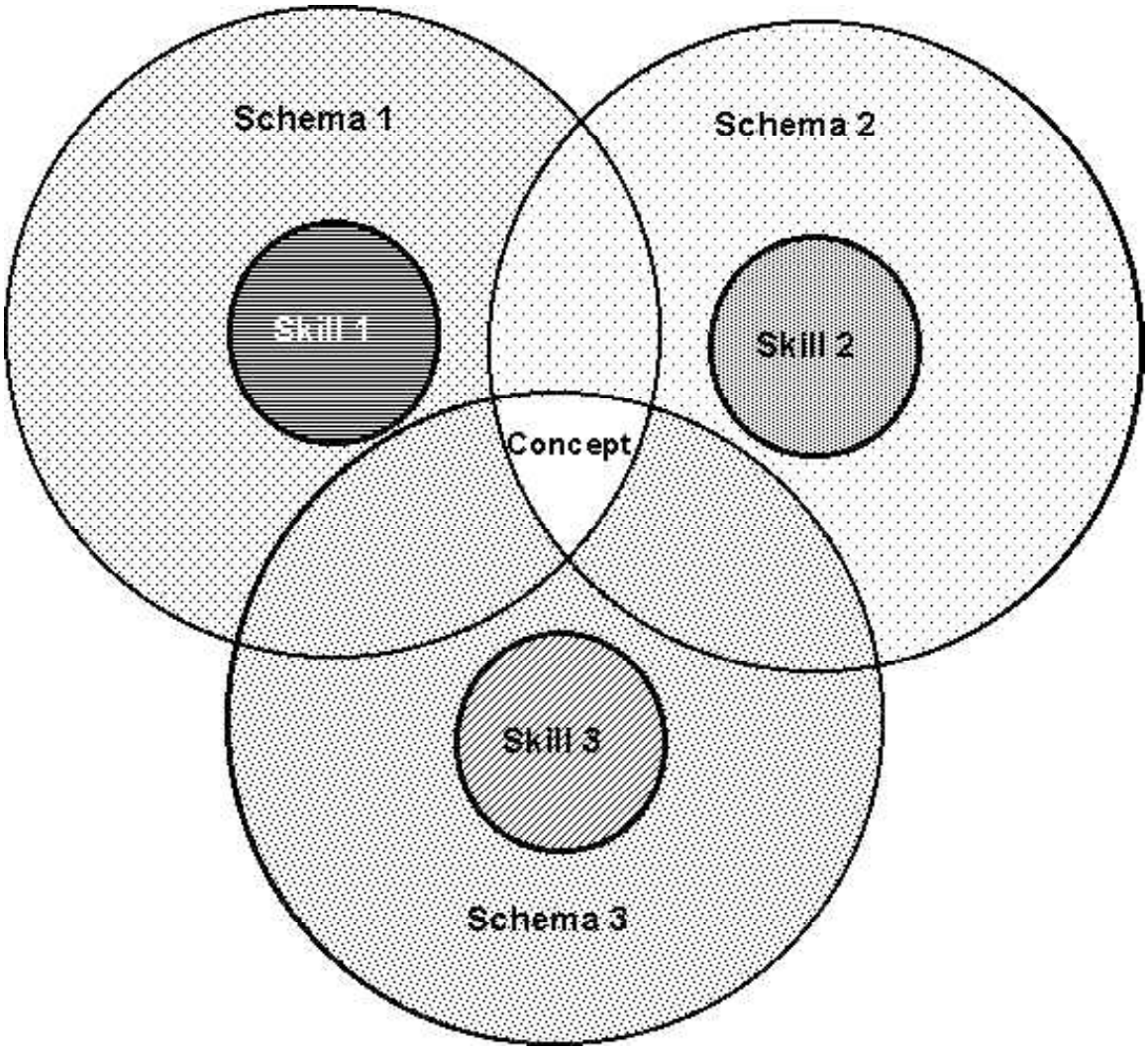


Figure 1 Concept map of relationship among skills, schema and concepts

Students learn Skill 1 (represented by the smaller circle) and build a mental schema representing a generalization of this particular skill (indicated by the larger circle), Schema 1. This schema may consist of accurate representations of the computing concepts along with misconceptions. Students then learn subsequent skills and build corresponding schemas. The intersection of these schemas triangulates on a more accurate representation of the concept . As learners understand the concepts underlying an increasing number of discrete skills, they reduce the cognitive load required to represent the knowledge, as compared with the requirements of storing an increasing number of discrete, unrelated skills . Ultimately, the students' conceptual understanding becomes rich enough to allow them to apply the concepts to independent problem solving beyond what is possible with solely procedural skills. This approach not only engenders improved conceptual development, but also enhances retention and transfer .

**Design of the Computing Concepts and Competencies Course**

In the summer of 1996, the course design team set out to apply these principles and create a new introductory CS course for non-CS majors. They conducted a series of hour-long interviews with the chairs and faculty representatives of the 67 client departments whose students would be taking this course to identify the computing competencies and skills the client departments considered important for their majors in their future courses and careers. The competencies that the client departments identified were consistent

with many of the principles of FITness; they wanted students to be able to use computers to solve a variety of different problems across a variety of domains throughout the curriculum and into their careers. Since computing systems and software are changing rapidly, the ability to transfer the concepts to new computing systems and problems was a significant objective. Although understanding computing concepts is critical to meet these goals, the traditional approach of teaching programming had been shown to be no longer viable. This meant that the design team had to adopt a new approach to teaching computing concepts.

**Inductive, spiral curriculum.**

The design team decided to take the inductive approach discussed above and shown in Figure 1. Critical to the success of this approach is maintaining the importance of computing concepts. Although developing a set of computing skills is a necessary condition for this approach, it is not sufficient. Instead, the design team structured the curriculum to present increasingly challenging problems that require students to synthesize the apparently discrete skills they are learning. This would help students begin to comprehend the utility and importance of understanding the computing concepts for the practical requirements of adapting to new and changing software.

**Multiple "tracks".**

After analyzing the client department needs, the design team identified a set of "core" competencies that appeared to be common across the university. These included a) the basic functions of an operating system; b) hierarchical file structures; c) computer networking (E-mail, the Web, distributed file systems); d) the ability search for and make sense of information from a variety of sources; and e) the ability to use word processing software to create research reports. Beyond the core competencies, various departments had different requirements. Some wanted their students to be able to create more elaborate reports and presentations; some wanted their students to be able to perform sophisticated data analysis; and some wanted their students to use computers for financial and business applications.

Given the different applications required by the various departments, the design team decided that a single, monolithic course would not meet the needs of all of the clients. While the underlying computing concepts that the design team identified are similar across domains (e.g., using parameters to allow the computer to take different actions depending on the values of parameters) the instantiation of these concepts varies across applications. For example, Web pages can be used to demonstrate parameters by exploring the effects of changing parameters in the HTML tags for Java applets. Students can use spreadsheets to see the effects of changing function parameters on the resulting calculations. Multiple tracks allow students to learn the underlying concepts in a context that is useful to their majors. This approach maintains student interest and motivation, a key factor for learning .

**Problem-based, collaborative learning.**

Since one goal was preparing students to use computers for problem solving in their domains, the design team decided that problem-based learning had to be a major component of the instructional design. Students had to actively use computers to solve problems rather than listening to lectures. This meant that all instruction had to take place in a computer laboratory so students could be working with computers continuously during class time.

Because this is an introductory course for which there are no prerequisites, incoming students have a wide range of computing knowledge and experience. This presents a number of instructional challenges such as a) ascertaining and keeping pace with changing incoming student experience; b) maintaining motivation among the more experienced students; and c) ensuring that novice students do not become discouraged or fall behind. The design team decided on a collaborative learning model to address these challenges. Each day the students are randomly assigned to groups of two to four so that they work with different peers every day. One day, a student may be the more knowledgeable other in the Zone of Proximal Development and will provide scaffolding for the less knowledgeable students in the group. The next day that student may be

the less knowledgeable member of the group and have to articulate to the other group members what is unclear or difficult about the particular problem.

Each class consists of a series of focal problems, the solution of which requires students to learn and practice new skills. Each problem builds on previous skills and concepts, extending the range of the students' capabilities (see Figure 1). Generally, students spend some time discussing possible solutions to the problem in small groups and then attempt to solve the problem. The TA then leads a debriefing where the groups report on their solutions and the problems they encountered. This helps the students focus on their problem solving and how it relates to the underlying concepts.

**Performance-based assessment.**

Assessment is a critical component of the instructional design. Because the instructional goals included retention and transfer, and the instructional design uses collaborative learning, the assessments needed to be criterion-referenced, rather than norm-referenced. Otherwise, students would view each other as competitors, undermining the goals of collaborative learning . Finally, to evaluate students' problem solving skills, the design team knew it was important that the assessments not be surrogate measures such as multiple choice tests but rather they wanted to use authentic, performance-based assessments.

To accommodate individual student differences, keep assessments consistent with the goals of encouraging student problem solving, and work within the university constraints of a fixed-credit semester, the course uses a modified mastery, performance-based assessment model. In this model, the curriculum progresses at the pace specified in the syllabus. At regularly scheduled intervals, students take a Bridge Task (BT) that requires them to synthesize the concepts and competencies to that point in the course. The students use their homework, in-class assignments and materials provided to them as part of the bridge task to solve the problem .

Each bridge task consists of a series of dimensions. Each dimension contains multiple instances. An instance is the smallest unit of text that can define a problem, sub-problem or a task that is representative of that dimension. For example, one dimension may address using reference tools. Within that dimension, there are several instances. One instance may require the students to use a thesaurus to look up a synonym for a particular word. Another may ask students to use the dictionary to insert a definition of a word into a document. The next dimension may address text fonts, with one instance of 10 point Times Roman, another of 12 point Helvetica, and so on.

Each bridge task is constructed by randomly selecting a single instance from each dimension and combining them so that each student receives a unique bridge task. The bridge task appears to the student as a long "story problem" in which the student has realistic scenarios of problems to solve. The creation, maintenance, delivery and record keeping needed to address these factors requires an elaborate, database-driven software infrastructure . The database also contains the scoring rubrics used by the raters to evaluate the students' performance on the BTs.

The pass/fail rates on each instance of each dimension can be analyzed to determine which instances have results that deviate from the target performance goals. If such items are found, they are revised or the instruction associated with the concepts these items test is revised. By analyzing the repeat rates for the bridge tasks the instructors can evaluate the overall difficulty and discrimination of the bridge tasks to determine if they are meeting the instructional and performance goals.

To test transfer, each bridge task contains one or more extension tasks that require students to apply the concepts and principles they have learned to solve new problems they have not previously encountered. Although learning a set of skills for using particular software may allow students to complete routine tasks, they must understand the underlying concepts or principles to complete the extension tasks.

Bridge tasks are evaluated on a mastery level pass/fail basis. If a student demonstrates sufficient mastery on the first bridge task, he or she "locks in" a grade of 1.0 in the course. If a student fails a bridge task, he or she must repeat the failed bridge task until passing before taking the next bridge task. For each subsequent bridge task passed, the student's course grade is incremented by 0.5 until he or she has passed the 3.0 bridge task. Once the student passes the 3.0 bridge task, he or she completes an integrative semester project that may increase his or her course grade to 3.5 or 4.0.

In traditional mastery learning, students continue to work on the course materials until they demonstrate mastery of specified materials at the desired level. They do not take a fixed set of examinations in order to receive a grade on the basis of single-attempt assessments . Instead of the instructor setting the pace, mastery learning can accommodate individual student variation . However, in a large university, students are expected to complete courses within a single semester, so the design team had to adapt the course to use a modified-mastery model. Students proceed through the course at the pace outlined in the syllabus. There is a maximum of twelve opportunities to take bridge tasks during the semester and a total of five bridge tasks that students must pass to be eligible to complete the final project. Therefore, students may take each bridge task two times and still be able to complete them all. Allowing students to repeat failed bridge tasks and providing twelve testing opportunities addresses the problems of inter-task reliability and generalizabilty in performance-based assessments .

There are several advantages to this assessment model. First, bridge tasks are criterion-referenced, not norm-referenced. While students complete the bridge tasks individually, they are not evaluated on a competitive basis; students' grades are not dependent on doing better or worse than their peers. Second, bridge tasks provide formative feedback, resulting in a greater opportunity for learning than traditional multiple choice examinations. The students' motivation shifts from accumulating points to mastering the concepts so they can complete tasks similar to those they will encounter in their subsequent courses and after graduation. Third, bridge tasks provide summative feedback; the course grade actually indicates which concepts and competencies a student has mastered. In courses that use norm-referenced assessment, a grade of 2.0 often means that a student has accumulated the mean number of points from a variety of exams and homework assignments. The grade does not indicate what knowledge the student does or does not have. In this model, the course grade reflects the concepts and competencies on which a student has demonstrated mastery.

**Course Implementation**

Institutional demand for the course necessitates an enrollment capacity of 1950 students per semester. The design goal of meeting each class in a computer laboratory coupled with the capacity of the computer laboratories required 65 sections of 30 students. Resource constraints dictated that each section would be met by teaching assistants (TAs) rather than instructors. There is a "lead TA," usually a graduate student, and an "assistant TA," usually an undergraduate student. To ensure instructional consistency across sections, the design team created detailed lesson plans, exercises and homework assignments for each day's instruction.

Each section of the course meets twice per week for one hour and 50 minutes per meeting. Each class consists of a series of problems on which the students work. Each exercise begins with the lead teaching assistant setting up the problem and leading discussions about how the concepts apply to the problem. The students next work for 5 to 30 minutes to solve the problem. During this time, both teaching assistants circulate among the students to facilitate the students' metacognition about their problem solving by asking leading questions such as "Where did you look in the help system?" or "How is this problem similar to (or different from) other problems on which we have worked?" After the students complete each exercise, the lead TA then guides a discussion of the problem, calling on students to review their solutions and asking questions to help the students to reflect on the relative merits of their solutions.

Homework assignments generally extend the material learned in class and set up material for use in the subsequent class. Homework is not collected and graded in the traditional sense. Rather, the homework is used in the next day's classroom assignments and is used to complete the bridge tasks.

## Evaluation of the Students' Conceptual Understanding

Since student outcomes are based on criterion-referenced, rather than norm-referenced assessments, it is not appropriate to assume a multivariate normal distribution of the variables. The bridge tasks are evaluated on a dichotomous basis, so the pass rate follows a logistic distribution . Therefore, we need to use multivariate techniques that do not require a normally distributed dependent variable. Discriminant analysis is a multivariate technique similar to multiple regression that identifies an optimal linear combination of independent variables that best discriminates among two or more groups. The resulting discriminant functions can be used to classify cases into the groups defined by the dependent variable.

Discriminant analysis is usually applied in three stages: 1) derivation of the discriminant functions, 2) validation of the discriminant functions and 3) interpretation of the resulting functions . In the derivation stage, the data are randomly divided into two sets. The first set of data is used to generate the best linear combination of the independent variables that predict the group membership.

The validation stage uses the computed classification scores for each individual to predict each individual's group membership. The number of individuals for whom the predicted group matches their actual group determines the accuracy of the prediction. Accuracy may be evaluated using standard chi-square expected frequency measures.

If the first two stages of discriminant analysis are successful, the third stage is interpretation of the resulting functions . This stage is similar to interpreting factor loadings in factor analysis. First, we examine the discriminant functions to determine the contribution of the independent variables to each function. After that, we attempt to characterize the differences among groups based on their multivariate means or group centroids. While factor analysis attempts to identify commonality among variables (factors) that underlie the observed variables, interpreting discriminant functions involves identifying common factors that are associated with individuals who are classified in the same group (dependent variable.) This process allows us to gain a better structural understanding of the BT skills and concepts that contribute to FITness. For a more extensive discussion of the analytic procedures see Urban-Lurain .

### Data Sources

The analysis was based upon data from students who took the course in fall, 1998, spring, 1999 and fall, 1999 (n = 5088). The students in this study represent a cross-section of undergraduates from non-technical majors at a large, public university. Their ACT scores are normally distributed with a mean slightly above the national average. Over half of the students are women. There are no gender differences in student outcomes in the course. The students are ethnically diverse; 81% are white, 10% are black; Asian students outnumber Hispanic, Chicano and Native American students. Black, Hispanic and Chicano students received lower grades in the course than do white and Asian students. However, ethnicity is not a factor after controlling for incoming academic ability (GPA and ACT scores) and student participation in the course. The students have a variety of incoming computing experience, but students' self-reported experience using computers before taking the course is not a factor in student outcomes.

### Variables

The final grade in the course was the dependent categorical variable used for the analyses. The university grades on a scale of 0.0, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5 or 4.0 for each course. The final course grades are shown in Table 1.

Table 1
Final Course Grades

| Course Grade | 0.0 | 1.0 | 1.5 | 2.0 | 2.5 | 3.0 | 3.5 | 4.0 |
|---|---|---|---|---|---|---|---|---|
| n | 83 | 191 | 345 | 629 | 1028 | 845 | 294 | 1673 |

Recall that the highest bridge task passed determines course grades up through 3.0. After that, students may raise their grade to 3.5 or 4.0 by completing a final semester project. Thus, all students receiving grades of 3.0 or higher passed the 3.0 bridge task by the end of the course. Because the bridge tasks and semester project are criterion referenced, we should not expect a normal distribution but rather a more linear one . Table 1 shows that the grade distribution is more linear than normal (M = 2.91, SD = 0.98, Skewness = -0.55). However, there is a "dip" at the 3.5 grade; most students who complete the final project receive full credit for it and earn a 4.0 in the course.

The independent variables were the number of times each student attempted each dimension on each bridge task. Table 2 shows an alphabetic list of the skills and concepts evaluated by dimensions on the 1.0, 1.5, 2.0 and 2.5 bridge tasks. These first four BTs evaluate the basic computing concepts. The 3.0 BTs focus on more advanced application of these concepts and are unique to each track. Not all students pass the 2.5 BT and attempt the 3.0 BT. Therefore, including the 3.0 BT and partitioning the population by track would prevent us from contrasting the results with the previous analyses in a meaningful manner, so only BT data through the 2.5 BT were included in the analysis.

Table 2
Bridge Task Skill and Concept Difficulty

| □ | Number of times did not pass dimension | | | | | | Did not take BT |
|---|---|---|---|---|---|---|---|
| Concept / Skill | 0 | 1 | 2 | 3 | 4 | 5+ | |
| Boolean search | 63.1 | 25.8 | 6.4 | 1.3 | 0.4 | 0.1 | 2.9 |
| Computer specifications | 45.9 | 27.9 | 11.7 | 3.9 | 0.9 | 0.2 | 9.4 |
| Create chart | 59.3 | 19.4 | 3.4 | 0.6 | 0.0 | 0.0 | 17.2 |
| Create link | 85.3 | 10.0 | 1.5 | 0.2 | 0.0 | 0.0 | 2.9 |
| Create private folder | 82.0 | 9.5 | 1.4 | 0.2 | 0.0 | 0.0 | 6.9 |
| Document margins | 73.2 | 16.2 | 3.1 | 0.4 | 0.1 | 0.0 | 6.9 |
| Extension task: backgrounds | 76.5 | 14.2 | 2.0 | 0.4 | 0.0 | 0.0 | 6.9 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Extension task: character styles | 70.4 | 18.6 | 3.5 | 0.6 | 0.0 | 0.0 | 6.9 |
| Extension task: Excel function | 62.7 | 16.9 | 2.9 | 0.3 | 0.0 | 0.0 | 17.2 |
| Find and rename file | 80.6 | 16.1 | 2.4 | 0.5 | 0.1 | 0.0 | 0.4 |
| Find new application | 88.9 | 9.4 | 1.1 | 0.2 | 0.0 | 0.0 | 0.4 |
| Footnotes in Word | 75.7 | 14.3 | 2.4 | 0.6 | 0.0 | 0.0 | 6.9 |
| Image on Web page | 83.2 | 11.6 | 1.9 | 0.4 | 0.0 | 0.0 | 2.9 |
| Modify styles in Word | 65.7 | 19.8 | 5.1 | 1.7 | 0.6 | 0.1 | 6.9 |
| New spreadsheet | 61.0 | 18.5 | 2.5 | 0.7 | 0.0 | 0.0 | 17.2 |
| Path to document | 71.2 | 17.8 | 3.4 | 0.5 | 0.1 | 0.1 | 6.9 |
| Path to own AFS | 59.3 | 31.2 | 7.0 | 1.6 | 0.4 | 0.2 | 0.4 |
| Public folder | 78.5 | 16.5 | 3.5 | 0.8 | 0.3 | 0.1 | 0.4 |
| Search for Web pages off-site | 80.2 | 14.2 | 2.3 | 0.4 | 0.0 | 0.0 | 2.9 |
| Table of Contents | 60.7 | 24.1 | 6.5 | 1.5 | 0.3 | 0.0 | 6.9 |
| Update payroll data | 47.3 | 26.0 | 7.5 | 1.7 | 0.2 | 0.1 | 17.2 |
| Web page text formatting | 88.4 | 7.5 | 1.0 | 0.2 | 0.0 | 0.0 | 2.9 |
| Web pages in Web folder | 88.1 | 7.8 | 1.1 | 0.1 | 0.0 | 0.0 | 2.9 |
| Web page URL | 52.3 | 31.5 | 10.0 | 2.5 | 0.6 | 0.1 | 2.9 |

Note: All numbers are the percentage of students who failed the dimension listed in each row the number of listed in the column headings.

The percentages of students who did not pass each dimension a particular number of times are listed in each column. If students never fail that dimension – however many times they take it – they are counted in the 0 column. If students do not pass a dimension one time, but then pass it on subsequent BTs – or never take the BT again – they are listed in the 1 column. Students who never took the BT associated with that dimension are listed in the Did not take BT column. For example, 63.1% of the students never failed the

Boolean search dimension. Some may have passed the BT and moved on; others may have failed that BT for other reasons, but never failed the Boolean search dimension when repeating the BT. Another 25.8% of the students failed this dimension one time, with a small number failing two, three or more times. Only 2.9% of the students never took the BT on which this dimension appeared. Because spreadsheets appear on the 2.5 BT, and some students never get to the 2.5 BT, 17.2% of the students are listed in the Did not take BT column for dimensions associated with that BT.

The dimension pass rate statistics are approximately logistically distributed, as expected for criterion-referenced items that are evaluated on a pass/fail basis. Since they are not normally distributed, we must transform them to use them as independent variables in discriminant analysis. Each of these pass rates was re-coded as a binary variable. If the student never failed the dimension, it was coded 0; all other students – including those who never took the BT – were coded 1.

**Derivation of discriminant functions**

The cases were divided into two groups. The first group had 70% of the cases and was used to derive the discriminant functions (n = 3038). The second group had the remaining 30% of the cases and was used for the validation stage. The independent variables were included and removed in a stepwise fashion, selecting variables that maximized the Mahalanobis distance between the closest groups . After 23 steps, the F test for inclusion and removal was no longer significant and 15 variables remained in the model. An orthogonal rotation of the discriminant functions maximized the loadings of the independent variables. After rotation, percentages of the variance accounted for by each of the function were: function 1 = 56.4%; function 2 = 21.5%; function 3 = 8.8%; function 4 = 6.1%; function 5 = 4.0%; function 6 = 2.1%; and function 7 = 1.1%.

**Validation of discriminant functions**

Because there are 8 groups, the analysis produced 7 discriminant functions that were used to classify the cases. Equal prior probabilities (0.125) were assigned to each group for the classification. The separate group covariance matrices of the discriminant functions were used for classification because Box's M test of equality for the matrices was significant.

The functions were validated by using them to classify the original sample (n = 3068) where they classified 40.2% of the cases correctly ($p < .000$). For cross-validation 39.1% of the "hold out" sample (n = 1355) were correctly classified ($p < .000$). Misclassification patterns were comparable in both samples, with the least accuracy around the 3.0 to 3.5 group boundaries.

**Interpretation of discriminant functions**

The seven discriminant functions define the multidimensional conceptual space in which the cases are clustered. To understand the structural relationships among the individual BT dimensions, we examine the correlations between the variables and the rotated functions shown in Table 3.

Table 3
Correlations Between Variables and Rotated Functions

| □ | Function | | | | | | |
|---|---|---|---|---|---|---|---|
| □ | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| Boolean Search | .067 | .013 | .443 | .489 | -.097 | -.094 | .332 |
| Create Excel chart | .637 | -.076 | .099 | .004 | -.018 | .106 | -.005 |
| Computer specifications | .205 | .145 | -.070 | .159 | .006 | .859 | .011 |
| Create private folder | .200 | .893 | -.154 | .164 | -.087 | .032 | .032 |
| Find and rename file | .033 | .006 | .170 | .012 | .269 | .345 | .395 |
| Excel function | .783 | -.041 | .036 | -.249 | -.020 | -.035 | -.060 |
| Find new application | .054 | -.011 | .090 | -.003 | .812 | .033 | .101 |
| Footnotes in Word | .287 | .536 | -.251 | .264 | .193 | -.132 | .116 |
| Modify styles in Word | .253 | .342 | -.369 | .307 | .340 | .140 | .296 |
| New spreadsheet | .727 | -.139 | .193 | .028 | -.061 | .016 | -.029 |
| Own web page URL | -.019 | .070 | .307 | .710 | .045 | .007 | -.075 |
| Table of contents | .230 | .291 | -.085 | .400 | .224 | .173 | -.511 |
| Update spreadsheet | .487 | -.193 | .291 | .091 | -.183 | .032 | .188 |
| Web page text formatting | -.020 | .418 | .726 | -.152 | .214 | .030 | -.001 |
| Web pages in web folder | -.052 | .324 | .797 | -.017 | .237 | -.021 | .004 |

We can "view" the correlations in Table 3 either along the rows or down the columns. If we look across the rows, we can see on which function, or functions, each variable loads most strongly. Some variables load primarily on a single function. For example, Computer specifications correlates primarily with function 6 ($r$ = .859) but does not have any correlations $r$ > .205 on other functions. This indicates that the concepts measured by the Computer specifications dimension (computer memory, secondary storage and their relationship to software specifications) are not strongly related to any other concepts in the students' minds. Other variables load moderately on multiple functions. Modify styles in Word loads somewhat on all seven of the functions with strength of the correlations ranging from $r$ = .140 on function 6 to $r$ = -.369 on function 3. This indicates that students' concepts associated with the Modify styles in Word are more diffuse. Modifying styles requires understanding styles as collections of formatting commands, what the formatting commands impact (character or paragraph attributes), when it is appropriate to modify styles rather than create new ones, and the abstraction of using styles rather than directly applying the formatting to particular sections of text.

By examining the columns in Table 3, we can see the students' conceptual grouping of the BT dimensions. Function 1 loads primarily on the concepts needed to solve more complex spreadsheet problems, as the largest correlations are with the extension task of using new Excel functions and the ability to create a New spreadsheet that solves a particular problem. Function 2 correlates most strongly with the dimension Create private folder, which requires understanding hierarchical networked file systems and the concepts of file permissions in order to create folders that are not accessible by other users. Function 3 correlates primarily with the HTML concepts needed to control Web page text formatting and understanding how Web servers access files as measured by the dimension Web pages in web folder.

## Discussion

Recall that Figure 1 represents the course's inductive approach to developing conceptual understanding with information technology. By having students work on problems that epitomize the important concepts, they develop mental schemas that triangulate on the concepts. The concept map in Figure 2 represents the students schemas and conceptual frameworks as measured by the function loadings in Table 3.

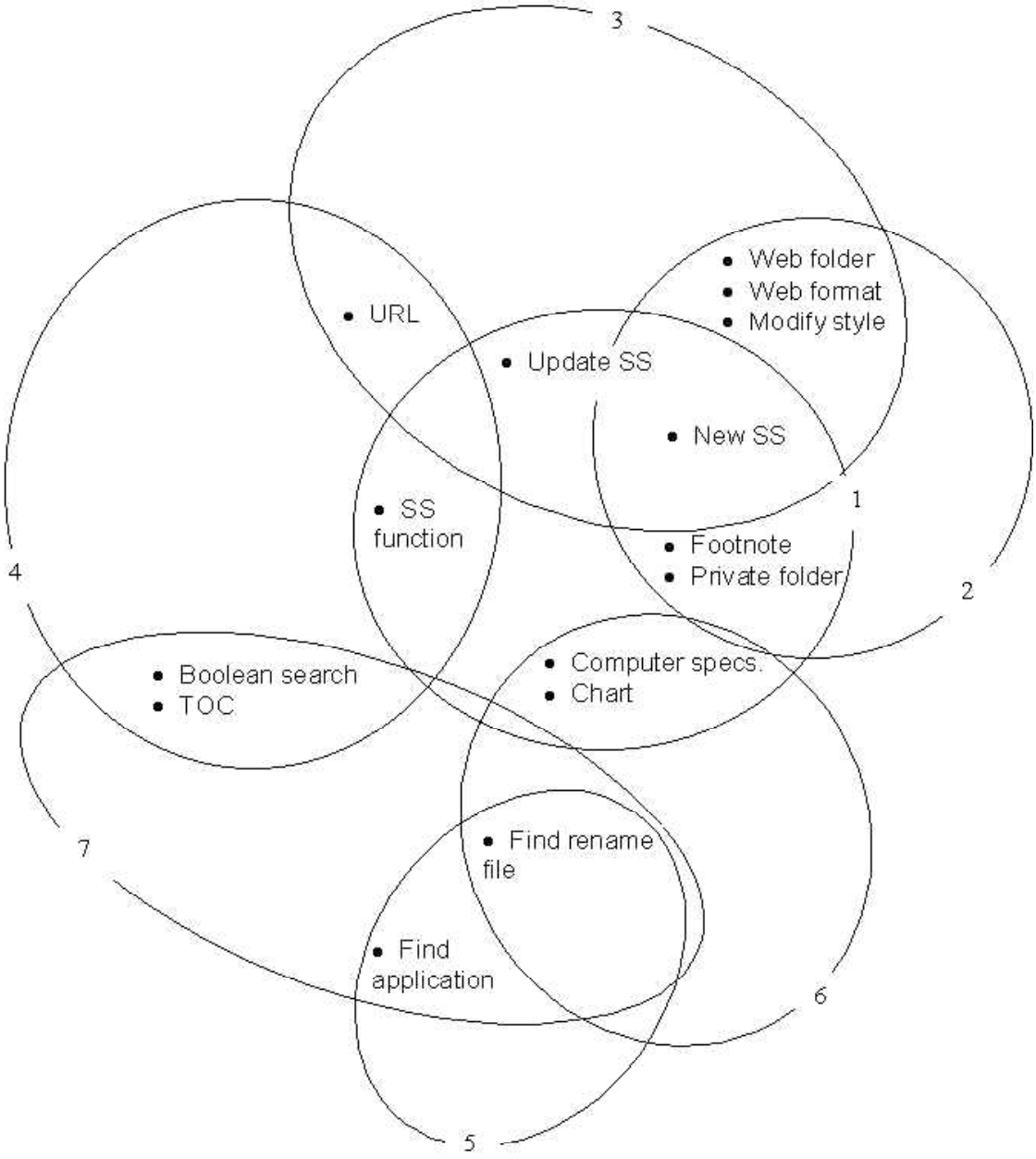


Figure 2 Concept map of computing skill schematic structure

is a two-dimensional depiction of the relationships among what are actually seven-dimensional ellipsoids. The volumes, shapes, and locations of these ellipsoids in seven-dimensional space are defined by the seven discriminant functions.

Each of the skills and competencies shown in the figure correlates – to various degrees – with each of the seven functions. However, to represent this mapping in two dimensions, each of the skills and concepts is located in the intersections of the ellipses representing the two or three functions with which it most

strongly correlates. For example, the BT dimensions for Footnotes, and Creating a private folder are each most strongly correlated with functions 1 and 2. The dimension for Footnotes also has moderate correlations with functions 4 and 5, but these higher dimensional intersections cannot be represented in two dimensions. The BT dimension New spreadsheet correlates with primarily with function 1 – and to a lesser degree – with functions 3 and 2. Function 1 is central in this model because seven of the 15 BT skills and concepts have their top two correlations with this function and it accounts for 56.4% of the variance in the model.

Each ellipse represents a different schema (cf., Figure 1). The intersection and clustering of the ellipses represents the way the students' schemas intersect and triangulate on the underlying computing concepts. The actual schematic structure captured by the discriminant functions is too complex to display in two-dimensions. However, the concept map does give a sense of the clustering of the schematic structure. The closer that a skill or competency lies to the middle of the graph, the more abstract and inter-related with other concepts it is. For example, Find and rename file and Find application both require knowledge of hierarchical file systems and appear near the outside of the graph. Private folder also requires knowledge of hierarchical file systems plus knowledge of networked file system permissions so it is grouped with data representation abstractions that are necessary to create footnotes, modify styles and control Web page formatting.

Several spreadsheet concepts also cluster together. Updating a spreadsheet is conceptually easier than creating a new spreadsheet. Updating primarily requires an understanding of the syntax of spreadsheet formulas. However, creating a new spreadsheet is more complex. It requires the ability to analyze a problem, determine the correct formulas to solve the problem and implement the solution by using the correct spreadsheet syntax.

**Instructional Findings**

This study shows that it is possible to teach FITness without teaching programming. However, the course's instructional design was a crucial part of its success. The instruction must be based upon an analysis of the critical concepts, their interrelationships, and the types of problems that epitomize these concepts at each stage of the learning process. For an inductive approach to work, the concepts need to be the right "depth" in relationship to the problems, about one "level" lower than the scope of the problem students are trying to solve. For example, logical operators are a critical computing concept that can be taught at many conceptual levels. One approach is that of many introductory deductive logic classes, teaching truth-functional composition, consistency trees and derivations. Another approach is teaching programming and the implementation of the underlying bitwise operators. Some courses go even lower, teaching about the implementation of logic gates in hardware . Although these are all valid representations of logical operations, knowledge of VLSI circuitry is not what is important when trying to locate the appropriate information in a vast sea of electronic databases. Translating a vague question into the appropriate set of keywords, combining the keywords with Boolean operators, evaluating the quality of results returned by the search – refining it to locate the relevant information – help the learner build an understanding of logical concepts that will meet the needs of FITness.

**Authentic, performance-based assessments.**

The final question this study explored is how to assess FITness. The course designers created unique, criterion-referenced, performance-based assessments – the bridge tasks – that are a key component of the modified-mastery model course design.

There are two broad categories of performance assessments. Task-centered performance assessments evaluate particular skills and competencies using clear-cut scoring rubrics. Construct-centered performance assessments emphasize general skills, but do not have clear-cut scoring guidelines and are more difficult to use for summative evaluation . The bridge tasks fall into the task-centered category, allowing for detailed scoring rubrics to ensure high inter-rater reliability. This study found that it is possible to create

sophisticated, problem-based, performance assessments on a large scale (over 40,000 BTs in this study) with scoring rubrics that produce grading error rates of less than 5%.

Inter-task reliability is another concern with a large number of complex performance measurements . Student performance can be affected by variability in the sampling of the particular tasks that they receive. The design of the bridge tasks addresses this problem. The granularity of the bridge task dimensions allows for analysis of the variability of each instance within the dimensions. The instructors use these analyses to reduce the variance within each dimension of the BTs. Students also have up to 12 opportunities to take BTs, so that their grades are not dependent upon a particular instance of a BT . These two factors allow the BTs to produce ratings that have high generalizability.

## Implications

The results of this study have implications for teaching FITness and for the use of technology in instruction in any number of disciplines.

### A Single Computing Course or Computing Across the Curriculum?

One of the goals of FITness is to prepare students to use information technology in all facets of their lives. This means that students will need to use computing technology as an integral part of their other courses, much as they use writing in many courses. As with teaching writing fluency, one debate about teaching FITness concerns teaching FITness in a single course versus "computing across the curriculum." As with the writing debate, there are arguments for and against both perspectives. This study has implications for this debate.

Computing across the curriculum has several advantages from the perspective of learning theory. First, it is contextualized, that is students learn how computing is used in a particular context while studying that discipline. This improves student motivation, which should enhance retention. On the other hand, there are several pragmatic disadvantages to this approach. As we have seen, FITness requires conceptual understanding. If computing across the curriculum is going to develop that conceptual knowledge, the computing aspects of the instruction must be designed to facilitate that conceptual understanding. This means that to design the instruction, faculty across domains must have a deep conceptual understanding of computing, much deeper than is required for a practitioner in the discipline to be FIT. Given that discipline-specific courses have as their primary focus the discipline content, it seems unlikely that faculty in all disciplines are going to have the time or conceptual knowledge of computing to design instruction that effectively builds conceptual understanding.

The advantages and disadvantages of a single computing course are the inverse of "computing across the curriculum." A single course can develop the material in depth and focus on teaching FITness as the primary task. It can provide uniform content so that all students have same foundation as they arrive in their subsequent courses. Because of the broad demand for FITness, a single course may have a high enrollment demand, so economies of scale can reduce the costs. On the other hand, there are many disadvantages to this approach. Foremost is that a single course isolates FITness from the other curriculum in the minds of the faculty who teach the course, the students who take the course and the faculty in other domains. Faculty who teach these courses usually develop them in isolation from the other curriculum, deciding what they think is appropriate to teach without knowledge of how students will use computing in their subsequent courses . Many students see such a course simply as a requirement, a hurdle to clear. If students do not see the relevance to their other courses motivation suffers. Faculty in other domains may not feel the need to integrate information technology into their courses because the students "had" a computing course. Finally, if students do not have opportunities immediately and continuously to apply what they learn in the computing course in other courses, retention suffers.

This study suggests two solutions to this dilemma. For institutions that chose to approach FITness with a single course, this study shows the importance of designing the course collaboratively among CS faculty

and faculty from other disciplines. The course designers must determine the contexts in which computing is used in the various disciplines to provide exemplar problems that can be solved using computing technology. From these problems, faculty can determine how to approach the problems so that they highlight the relevant computing concepts. Doing so will produce a course in which the students solve problems that are situated in different domains but epitomize the underlying computing concepts. Crucial to this approach is the cooperation between the CS and other discipline faculties to ensure that the course's problems are relevant to the domains and that the subsequent discipline courses build upon the problems and concepts from the course.

For institutions that prefer to implement computing across the curriculum, the same principles can apply. In this context, the CS faculty can work also with faculty from other domains. The discipline faculty can provide problems within the discipline and their typical solutions. The CS faculty can work with the discipline faculty to design instruction that highlights the computing concepts and demonstrates using these concepts to solve the problems. This approach will require allocating sufficient time within the curriculum so students are able to learn the underlying computing concepts by struggling with the problems. It will also require the discipline faculty to treat computing as an important part of the curriculum, not simply as a "tool" to be used to accomplish an end.

**The Importance of FITness**

This study suggests that – at least for the foreseeable future – higher education will need to address FITness. Some argue that it is not going to be necessary to continue to teach FITness because, as computers become ubiquitous, students will arrive at college FIT. This argument goes back to the 1980's . However, this study found that students are not arriving at college any more FIT than they did in the 1980's. Students have a great deal of exposure to using computers, but they have little conceptual understanding when they come into the course. As a result, they have little ability to apply technology to the solution of new problems in ways that technology uniquely enables.

One reason that that we cannot yet assume students will arrive from high school FIT is because FITness is contextual. FITness for K-12 students is not necessarily sufficient for college students. Students need to use computing technology to solve problems in domains that they are studying in college and will need instruction that focuses on problems that are typical of those they will encounter in their college courses.

A second implication for teaching FITness is that it requires conceptual understanding which will not emerge from a superficial training on particular features of applications software. Teaching for FITness must focus on problem solving, with an emphasis on transfer, so students will be able to keep abreast of rapidly evolving technology and use it to solve new problems in new ways.

The impact and use of computing technology – and the definition of FITness – evolves differently across disciplines. Originally, computing was restricted to large numeric problems in business or science. Business students used to learn COBOL programming, not with the expectation that they would actually program in their business careers, but so that they understood what was involved in creating business reports with computers. Now, business students must do modeling with spreadsheets that requires an understanding of both the computing tool –spreadsheets – and the business problems that they are trying to solve. Spreadsheets are also transforming how students in technical and scientific disciplines use computers. Disciplines that once required FORTRAN programming to do data analysis now do much more sophisticated data analysis with spreadsheets and databases. Again, this requires both knowledge of the tool and knowledge of how to use the tool to solve the problem at hand in a particular discipline.

Today, disciplines that had not previously used computers are continuously finding new ways to use them. A few years ago, "innovative" teachers used computers for administrative tasks, such as tracking student grades in a spreadsheet. Now, computing is infusing all areas of the K-12 curriculum. To be FIT, teachers must understand computing concepts well enough to be able to find innovative ways of helping students learn with – and about – these technologies.

## References


Anderson, R. C. (1984). The architecture of cognition. Cambridge, MA: Harvard University Press.

Baker, F. B. (1985). The basics of item response theory. Portsmouth, NH: Heinemann Educational Books.

Biermann, A. W., Fahmy, A. F., Guinn, C., Pennock, D., Ramm, D., & Wu, P. (1994, March, 1994). Teaching a hierarchial model of computation with animation software in the first course. Paper presented at the Twenty-Fifth SIGCSE technical symposium on computer science education, Phoenix, AZ.

Block, J. H., Efthim, H. E., & Burns, R. B. (1989). Building effective mastery learning schools. New York: Longman.

Bloom, B. S., Madaus, G. F., & Hastings, J. T. (1981). Evaluation to improve learning. New York, NY: McGraw-Hill.

Bruner, J. S. (1960). The process of education. New York: Vintage Books.

Chandler, P., & Sweller, J. (1991). Cognitive load theory and the format of instruction. Cognition and Instruction, 8(4), 292-332.

Committee on Information Technology Literacy. (1999). Being fluent with information technology (Book and web site NSF Contract Number CDA-9616681). Washington, DC: National Academy of Sciences.

Foshay, R. (1991). Sharpen up your schemata. Data Training, May, 18-25.

Hair, J. F., Jr., Anderson, R. e., & Tatham, R. L. (1987). Multivariate data analysis with readings. (2nd ed.). New York: Macmillian Publishing Company.

Johnson, D. W., Johnson, R. T., & Smith, K. A. (1991). Active learning: Cooperation in the college classroom. Edina, MN: Interaction Book Company.

Keller, J. M. (1983). Motivational design of instruction. In C. M. Reigeluth (Ed.), Instructional-design theories and models: An overview of their current status (pp. 383-434). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Khattri, N., Reeve, A. L., & Kane, M., B. (1998). Principles and practices of performance assessment. Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Lee, J. F., & Pruitt, K. W. (1984). Providing for individual differences in student learning: a mastery learning approach. Springfield, Ill.: C. C. Thomas.

Levine, D. U. (1985). Improving student achievement through mastery learning programs. San Francisco: Jossey-Bass.

Mardia, K. V., Kent, J. T., & Bibby, J. M. (1979). Multivariate analysis. London: Academic Press.

Parker, J. D., & Schneider, G. M. (1987, February, 1987). Problems with and proposals for service courses in computer science. Paper presented at the Eighteenth SIGCSE technical symposium on computer science education, St. Louis, MO.

Raymond, M. R., & Viswesvaran, C. (1993). Least squares models to correct for rater effects in performance assessment. Journal of Educational Measurement, 30(3), 253-368.

Reigeluth, C. M., & Stein, F. S. (1983). The elaboration theory of instruction. In C. M. Reigeluth (Ed.), Instructional-design theories and models: An overview of their current status (pp. 338-381). Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Rodriguez, R. V., & Anger, F. D. (1981, June, 1981). A novel approach to computer literacy for natural science students. Paper presented at the National Educational Computing Conference, North Texas State University, Denton, Texas.

Shavelson, R. J., Baxter, G. P., & Gao, X. (1993). Sampling variability of performance assessments. Journal of Educational Measurement, 30(3), 215-232.

Shuell, T., J. (1986). Cognitive conceptions of learning. Review of Educational Research, 56(4), 411-436.

Smith, J. P., III, diSessa, A. A., & Roschelle, J. (1993). Misconceptions reconceived: A constructivist analysis of knowledge in transition. The Journal of the Learning Sciences, 3(2), 115-163.

Tennyson, R. D., & Cocchiarella, M. J. (1986). An empirically based instructional design theory for teaching concepts. Review of Educational Research, 56(1), 40-71.

Urban-Lurain, M. (2000). Teaching for fluency with information technology: An evaluative study. Unpublished Ph.D. Dissertation, Michigan State University, East Lansing, MI.

Urban-Lurain, M., & Weinshank, D. J. (1999a, March 26, 1999). "I Do and I Understand:" Mastery model learning for a large non-major course. Paper presented at the Special Interest Group on Computer Science Education, New Orleans, LA.

Urban-Lurain, M., & Weinshank, D. J. (1999b). Mastering computing technology: A new approach for non-computer science majors . http://aral.cse.msu.edu/Publications/AERA99/MasteringComputing.html.

Urban-Lurain, M., & Weinshank, D. J. (2000, October, 2000). Is there a role for programming in non-major CS courses? Paper presented at the Frontiers in Education 2000 Conference, Kansas City, MO.

Vygotsky, L. (1978). Mind in society: The development of higher psychological processes. Cambridge, MA: Harvard University Press.

Yelon, S. L. (1996). Powerful principles of instruction. White Plains, NY: Longman.