

NOTE TO USERS

This reproduction is the best copy available.

UMI[®]

Impact of a Pre-Programming Course in a Computer Science Curriculum

Robert James Faux

ID 109462

December 6, 2003

**Project Demonstrating Excellence
(Dissertation)**

**Submitted in partial
fulfillment of the requirements for the degree
Doctor of Philosophy in Interdisciplinary Studies
with a specialization in Computer Science Education
at The Union Institute and University**

UMI Number: 3160272

Copyright 2005 by
Faux, Robert James

All rights reserved.

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 3160272

Copyright 2005 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

Table of Contents

1	Problem Statement and Context.....	4
1.1	Abstract	4
1.2	Project Summary	5
1.3	Research Question	12
1.4	Scope and Limitations	14
1.5	Original Contribution	20
1.6	Social Relevance	21
2	Intellectual Context	23
2.1	Summary of Topic Scope	23
2.2	History of Curricular Change in Introductory Computer Science	25
2.3	Introduction to Computer Science.....	27
2.4	Related Computer Science Education Research.....	34
2.5	Context within Body of Knowledge	42
3	Methods	45
3.1	Research Design: Study Structure.....	45
3.2	Research Design: Sample	58
3.3	Research Design: Variables	64
3.4	Research Design: Data Collection Tools	76
3.5	Responsibilities of Participants	89
3.6	Review Processes	91
4	Findings	93
4.1	General Observations.....	93
4.2	Demographics	96
4.3	Pretest and Posttest Data.....	107
4.4	Self-Evaluation Data	118
4.5	CS0 Satisfaction Data.....	124
4.6	Qualitative Data	125
5	Interpretations and Analysis.....	131
5.1	Sample Representation of Population	131
5.2	Analysis of Programming Skill Learning	134
5.3	Analysis of Self-Evaluation Skills.....	142
5.4	Analysis of CS0 Satisfaction Levels.....	147
5.5	Data Mining Analysis	151
5.6	Qualitative Analysis	168
5.7	Interpretation of Findings.....	172
5.8	Summary of Findings	176
5.9	Recommendations for Future Research.....	180
6	Bibliography	182

Appendices

- I. New Curriculum Materials
- II. Demographics Data Collection Tool
- III. Exit Survey Data Collection Tool
- IV. Pretest Data Collection Tool
- V. Posttest Data Collection Tool
- VI. Informed Consent Form
- VII. Instructor Information
- VIII. Union Institute & University Institutional Review Board Materials
- IX. Minnesota State University (MSU) - IRB Materials
- X. Bemidji State University (BSU) - IRB Materials
- XI. BSU Data
- XII. Data Mining Samples

1 Problem Statement and Context

1.1 Abstract

Impact of a Pre-Programming Course in a Computer Science Curriculum

The value of integrating problem solving, algorithm development, algorithm testing, pseudocode, and diagramming techniques into introductory computer science courses has not been quantified in past research. It was hypothesized that the introduction of problem solving and algorithm development topics prior to the introduction of a programming language would reduce the learning curve requirements and increase the success rate for beginning programmers. Existing research suggests that advance organizers may aid learning tasks; this research seeks to confirm whether the addition of algorithm development concepts to the introductory curriculum serve as appropriate organizers for programming learning.

Supplementary materials were developed and used in a typical breadth-first, introductory computer science course (CS0). Weak treatment (control) and strong treatment (treatment) cohorts were tracked through their experiences in the subsequent programming course (CS1). The data collected in the CS1 course followed a standard treatment-posttest-posttest study format, with the treatment occurring in the CS0 course. Baseline problem solving skills, demographic information, and satisfaction ratings were collected at the beginning of the CS1 course and were paired with programming skill and satisfaction ratings at the end of that course.

Quantitative data analysis revealed that posttest programming scores for like tests exhibited no significant difference within the sample. However, there was an observed difference with the treatment group performing better than the control group. Specifically, members of the treatment

group reported a better experience than the control in the introductory (CS0) course, with significantly different results between groups on a standard Lickert scale. Similarly, the treatment group expressed opinions in qualitative evaluations that supported the conclusions suggested by scaled results. Treatment group members used pseudocode more consistently and applied diagramming concepts to aid in writing code during their posttest experience. Triangulation provided consistent results, which indicated that treatment group participants successfully applied algorithm development processes as part of a programming strategy. In contrast, control group members cited shortcomings in the original introductory curriculum that were directly addressed by the new approach. This confirms experiential and anecdotal evidence that suggests that prior learning of algorithm development skills can have a positive impact on programming learning.

1.2 Project Summary

1.2.1 Introduction

There is no consensus among computer science educators concerning what approaches serve students best in early learning stages (Walker and Schneider, 1996). Current guidelines for the introductory sequence in computer science programs provide multiple models for curriculum development (Computing Curriculum, 2001), which reflects this uncertainty. Unfortunately, the current body of knowledge in computer science education is relatively new and largely incomplete (Clancy, Stasko, Gudzial, *et al.*, 2001). Other than a growing source list outlining curricular innovations, supporting arguments, and anecdotal results, there is insufficient data to support any particular approach as a 'best practice.'

A corollary to the introductory sequence is how computer science disciplines can best support programming learning within the curriculum. It has become clear that current approaches are not consistently

successful in developing capable programmers, despite evidence that many curricula have been modified and simplified in an attempt to facilitate learning for a broader base of students (Tucker, Keleman and Bruce, 2001). Thus, there is a need for research to help establish processes that simplify the learning curve in regard to programming without lowering expectations for achievement. It is equally important that any efforts to support programming learning avoid perpetuating the myth that programming and computer science are one and the same (Powers and Powers, 2001). A successful approach should reflect the reality that programming is simply one of a broad set of tools in the discipline.

This project focused on the integration of beginning problem-solving and algorithm development techniques into introductory computer science pre-programming (CS0) courses. It was hypothesized that there would be a measurable improvement in student learning during subsequent first programming (CS1) courses as a result of the successful implementation of the new CS0 curriculum. The content and structure of similar pre-programming courses at Bemidji State University (BSU) and Minnesota State University at Mankato (MSU) were modified with supplementary problem-solving, pseudocode, diagramming, teamwork, and algorithm testing units. The effects of this change on learning in the subsequent first programming course was investigated in conjunction with the curricular alterations. The sample population consisted of students attending CS0 and CS1 courses offered by the Computer and Information Sciences departments at these schools during the three-term duration of the study. Students who attended the CS0 course prior to curricular alterations comprised the control group. The treatment group consisted of students who attended the course after alterations were made.

1.2.2 Study Goals

The main purpose of this research project was to measure the impact isolated curricular content change in the introductory (CS0) course had on the first programming course (CS1). General works on cognition and learning suggest that the identification and use of proper advanced organizers can provide significant support for future learning in any given discipline (Ausubel, 1968). The curricular changes selected were believed to be key concepts that provide advanced organizers for and promote success in programming, as well as in general computer science problem-solving. A candidate set of advanced organizers was used to modify the CS0 curriculum, and these changes were isolated by limiting other curricular changes between the control and treatment groups. Although it is possible that this set of organizers could benefit a broad range of computer science learning topics, the scope of this study was limited to the impact these changes had on programming learning. The intent was to determine whether students were more or less likely to achieve success in implementing programming tasks as a result of the emphasis on specific advanced organizers in the curriculum. Participants in the programming course completed a problem-solving pretest at the beginning of the CS1 course to provide baseline scores intended to measure incoming ability levels. A programming posttest was completed at the end of the CS1 course to measure the actual level of programming skill attained by the end of this course. The results of these tests provided an important data set for determining the success of the newly modified curriculum in supporting programming learning.

Evidence of success in learning is very difficult to measure and can suffer from the subjective nature of evaluation. In addition, numerous external variables impact learning in field research in education. Therefore, corroborating evidence was gathered to supplement the results gathered

in the testing process. Components of programming success (as paired with baseline ability), CS0 course satisfaction, self-capability ratings, and problem-solving approaches employed on the tests were used to triangulate the data collected.

Students' satisfaction with a course, learning event, or topic area can have an impact on the amount of learning that occurs in that course and in subsequent, related courses (Ben-Ari, 2001). Therefore, data were collected on general satisfaction, subject importance, and students' perception of the applicability and/or usefulness of the CS0 course. This was measured by collecting qualitative satisfaction statements and ratings based on a traditional Lickert-type scale at the beginning and the end of the CS1 course. A particular focus was the student's perception of how CS0 and CS1 related to each other, since the relationship between the introductory course and programming learning was the focus of the research. It was believed that significantly different responses between the control and treatment groups would constitute an indication that the curricular changes had some impact in providing an overall framework for the CS0 course that enhanced future learning.

Since students' perceived comfort with a subject and their expectations for success are often what will foster actual success or failure (Bay and Daniel, 2003), data were gathered with respect to each participant's comfort level in the areas of computing, mathematics, problem-solving, and programming. Information on these areas was collected as ratings on a Lickert-type scale at both the beginning and the end of the CS1 course. The first three ratings provided a baseline, which was not expected to change over the time interval or across groups. The programming rating was expected to change for both groups in response to the content of the CS1 course; however, data were collected to determine if there was a

significant difference in self-reported improvements in ability between the control and treatment groups.

Finally, qualitative observations were made to analyze the approaches used for problem-solving on both the pretest and the posttest. Participants were encouraged to 'show their work' on both tests in both groups. This additional work was coded in order to identify trends in approach and determine if there were correlations between levels of success and particular methods. Evidence that advanced organizers from the CS0 course were being used was expected to provide confirmation that a connection between the organizer and the targeted learning event had been made.

1.2.3 Summary of Curricular Alterations

Although the broad coverage of a CS0 course in a breadth-first model intentionally avoids devoting too much time to any one subject, it is the researcher's opinion that problem-solving and algorithm development provide an important foundation for much Computer Science subject material and study. Therefore, the curriculum for the treatment CS0 group was modified to strengthen and emphasize this topic area.

Many subject areas require problem-solving skills, but few require competence in this area to the degree that Computer Science does (Cook, 1997). The development of algorithms and their implementation in a programming language, in particular, relies heavily on problem-solving techniques (Ramalingam and Wiedenbeck, 1998). Since many students who enter post-secondary Computer Science programs have had minimal background and training with problem-solving techniques, coverage of problem-solving in the CS0 course curriculum was expanded. General problem-solving skills were included to provide a foundation for the development of more specific computer science problem-solving skills.

One common method for expressing the form of an algorithm in a manner that does not rely on a programming language structure is pseudocode. The existing CS0 courses defined the concept of algorithms and illustrated pseudocode examples for the student; however, algorithms development was not addressed much beyond this. The revised curriculum expanded the focus on pseudocode, encouraged the application of pseudocode use, and included a section on testing algorithms once they are written. The addition of testing was intended to encourage reflection once the student developed a candidate algorithm for use.

Diagramming techniques are one tool that provides students with a chance to visualize an algorithm (Naps, Rossling, Almstrum, *et al.*, 2002). In fact, pictures or diagrams frequently provide visual stimuli that support problem-solving, which is not the case with pseudocode. Simple diagramming techniques such as flow charts can supplement problem-solving, algorithm development, and pseudocode learning by giving students another tool. The inclusion of diagramming techniques in the CS0 curriculum was intended to increase the breadth of instructional methods and make the topic accessible to a wider range of students.

Collaboration is an important aspect of problem-solving, so team development often becomes a vital component of the undergraduate degree program (Powers, 2002). The team development section was included in the enhanced curriculum to promote successful collaboration in problem-solving, algorithm development, and testing. Furthermore, the emphasis on collaboration was intended to foster an appreciation of the importance of communicating ideas and designs to others.

1.2.4 Summary of Study Approach

The experimental design for this project represented a primarily exploratory piece of research for which a hypothesis, two subordinate hypotheses, and their corresponding null hypothesis were evaluated. This was a self-control study; participants were not randomly assigned to groups, and the control and treatment groups were not measured concurrently. The research was longitudinal in nature, with the progress of participants being monitored for a term and the contact with each group lasting through one school year (two academic terms). Descriptive research techniques were used in the demographic and exit survey data collection tools to provide evidence pertaining to the validity and reliability of data collected in this project.

This study follows a treatment – posttest – posttest design that tracks the progress of two cohorts during their attendance in CS0 and CS1 courses at the subject post-secondary schools. The first posttest is referenced as a pretest in this document because it occurred at the beginning of the CS1 course, in spite of the fact that it was given after the CS0 course. This design was derived from the base structure for nonequivalent control group designs provided by Cook and Campbell (1979). According to their definitions, this study constitutes a weak treatment versus a strong treatment structure, since both sets of individuals did receive some instruction about algorithms and problem-solving. Similarly, the groups will be called control and treatment groups for clarity and consistency.

In an effort to address external variables, data triangulation was employed. Data were collected on achievement, self-comfort ratings with related tasks, course satisfaction and applicability data, pertinent demographic data, and qualitative observational data from both achievement tests. Analysis triangulation was accomplished by combining

traditional paired analysis statistical methods, data mining methods, and qualitative analysis approaches.

There was no control over the selection of members for the treatment or control group; however, the structure of the study precluded a purely random selection of members in each group. Students were not informed that the content of the CS1 courses differed, and there was no concurrent difference in the CS0 courses. Since the control group had completed the CS0 course in an academic term prior to the treatment group, each group represented an entire class of students moving through the computer science curriculum for a given period of time. This followed a typical cohort design, except that it was possible for members of the control group to attend the CS1 course with treatment group members if they delayed continuation of their studies by a term. Students fitting this description were not included in the research population. In order to determine if the groups were representative of the sample population, demographic information was collected and compared to school and discipline norms.

1.3 Research Question

Will the integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course impact the learning of programming skills and application of problem-solving skills in the first programming course for post-secondary institution students?

1.3.1 Population

The purpose of this research project was to determine whether the integration of a set of beginning algorithm development concepts and problem-solving techniques at the beginning of a computer science curriculum would support improved learning for college-aged students. In this document, college-aged students were defined as all persons who were at least 17 years old. Based on school admission policies,

participants in these courses were presumed to hold a high school diploma or equivalent and to possess sufficient knowledge of prerequisite material. Terminology and assumptions for post-secondary students were based on the academic model utilized in the United States, although it seems reasonable to map terms to school systems in other countries. The target population consisted of persons interested in Computer Science courses and learning. In particular, this research focused on students in CS0 and CS1 courses in typical four-year computing degree programs. The sample population from which the study sample was drawn was the subset of students enrolled in post-secondary classes at the Bemidji State University (BSU) and Minnesota State University at Mankato (MSU) campuses. The study sample consisted of registered students in courses with participating instructors.

1.3.2 Curricular Modifications

The set of concepts and techniques integrated into the pre-programming course was intended to introduce students to the basics of algorithm development and problem-solving. These topics were selected because they were expected to have a direct impact on future learning in computer science and, in particular, programming learning. Treatments of diagramming, pseudocode, team development, algorithms, test plans, documentation, requirements gathering, control structures, variables, abstraction, and modularity were included in the pre-programming course. The control group had exposure to pseudocode, algorithms, modularity, control structures, and variables. For the treatment group, the existing topics were extended or refocused to include the other areas of concentration (diagramming, algorithm testing, and team development, as well as requirements gathering, documentation, and abstraction for problem-solving).

1.3.3 Measurements of Learning

The primary targeted learning skill measured in this research project was programming competence. The posttest provided participants with an opportunity to exhibit the skills they had acquired at the end of the CS1 course. In order to take prior skill levels into account, a pretest was used to collect baseline information that could be paired with the posttest data. In addition to these data collection tools, participants were asked to complete a self-evaluation that addressed problem-solving, computing skills, math skills, and programming skills. This information was used to corroborate the information provided by their performance on the tests. These data points were further supplemented by measurements of satisfaction for the CS0 course, and qualitative information captured through questionnaires given at both the beginning and the end of the CS1 course.

1.4 Scope and Limitations

Curricular modifications were isolated as much as possible in order to determine their impact on early programming skill development. Although findings here do not provide unqualified support for the contention that CS0 modules are inherently critical for success in CS1 modules, incidental support for this contention is warranted. This research supports the development of a more effective CS0 curriculum for programs that adopt a breadth-first strategy for early Computer Science learning. It also encourages those educators who advocate a breadth-also or depth-first approach to consider expanding coverage of these concepts prior to actual programming learning.

1.4.1 Curricular Change Limitations

In order to determine candidate topics for inclusion in the modified course, the existing curriculum for both participating institutions (BSU and MSU)

were evaluated. The course structure for both programs was based upon materials found in Schneider and Gersting's *An Invitation to Computer Science* (1999). Chapters one through five and chapter fourteen of this textbook constituted the framework for teaching and learning during the semester for both programs. These chapters included introductions to algorithms, pseudocode, control structures, algorithm efficiency, hardware concepts, and ethics in computing. All topics were addressed in a manner that was exclusive of programming languages. Faculty at BSU and MSU considered mastery of these concepts to be critical for success in their computing majors and minors; therefore, these topics could not be removed, nor could the time allotted to these subjects be reduced. Participating instructors agreed that three to four weeks of the fifteen-week semester were available for additions to the core content (outlined above) for this course.

1.4.2 Curricular Modifications

The existing course was modified by adding beginning problem-solving techniques, diagramming methods, team development, testing algorithms, and some rudimentary software engineering concepts to existing course topics. Algorithm and pseudocode discussion in the existing pre-programming course was supplemented with diagramming techniques for problem-solving and algorithm development. Although numerous formal diagramming techniques exist, students were exposed to informal diagramming processes that used flow charts and structure charts as problem-solving tools. A section dedicated to test plans for algorithms was developed to provide an appropriate transition from algorithm development to algorithm efficiency. This promoted the use of project scope identification and early problem discovery in the solution development process.

Diagramming methods and test plans for algorithms further emphasized the importance of algorithm development as a vital aspect of computer science. A segment on team development and team member roles provided a useful supplement to the professional ethics portion of this course and was created to support positive collaboration in computer science learning. Finally, a small additional segment provided students with an introduction to the basic concepts of software engineering, which tied the previous concepts together into the 'big picture' of software development. Topics in chapter seven of the text (Schneider and Gersting, 1999) were utilized for this segment, but the material was modified to exclude programming specific content. Software engineering topics included abstraction, modularity, requirements gathering, and documentation.

In summary, the curriculum changes to the pre-programming course were:

1. Problem-solving Techniques
2. Diagramming techniques
3. Test plans for algorithms
4. Abstraction and modularity concepts
5. Collaboration methods

1.4.3 CS1 and Program-Wide Curricular Framework

The first programming course at MSU and BSU was not altered in content. The existing programming courses utilized the Java language (MSU) or the C++ language (BSU) and included topics from beginning programming to intermediate data structures and programming. Instructors at both colleges indicated that no significant changes (other than normal semester-by-semester or instructor-by-instructor differences) occurred in the CS1 course during the time frame of this research project. Both programs utilized the breadth-first model for the introductory computer

science curriculum, and both had moved from a depth-first model at least two years prior to the inception of this study.

1.4.4 Data Collection Points

The relative success of students in the CS1 course based on their exposure to either the non-modified or the modified CS0 curriculum in the pre-programming course was evaluated. In other words, the treatment for this research occurred in the pre-programming course and data collection occurred in the programming course. In order to facilitate data collection, instructors were asked to implement two data collection instruments on the first day of the programming course. Data points collected at this time included basic demographic information, students' initial self-evaluation of skill, initial evaluation of the CS0 course, and a pretest used to determine a problem-solving baseline for students in the course. In addition, instructors were asked to implement two additional data collection tools near the end of their course. These data points included new evaluations of skills, new evaluation marks for the CS0 course, and posttest results that assessed programming skills. All test materials were implemented separately from graded materials, and students were informed that these results had no impact on their standing in the course. All completed measurement tool materials were submitted to the instructors, who then forwarded the unprocessed materials to the researcher.

In summary, the data collection devices utilized in the programming course were:

1. Demographic/Entry survey (beginning of CS1)
2. Pretest questions (beginning of CS1)
3. Posttest questions (end of CS1)
4. Exit survey (end of CS1)

1.4.5 Presentation of the Modified Curriculum

Curriculum modifications consisted of written materials in the form of lecture notes, exercises, and examples that were made available to CS0 instructors. Students were also able to view all materials via an online website. These materials occasionally included suggested activities or group tasks; however, the method of presentation of the new or existing course materials by the participating instructors was not dictated. The inclusion of these new materials and modifications represent the full extent of control exercised on the instructors, who were otherwise free to choose their own instructional methods. The instructor remained the same for both the control and treatment groups in the CS0 course, which lessened the impact of instructional differences.

Instructors differed in the CS1 courses, although it was understood that the curriculum in that course would remain the same for each section. These instructors were only asked to perform data collection tasks during the term. No effort was made to encourage faculty members to alter their teaching in this course, nor was data collected about their teaching styles or competence.

1.4.6 Participating Schools

Two separate entities, Bemidji State (BSU) and Minnesota State at Mankato (MSU), were involved in this research project. The researcher was not affiliated with either organization during the project duration, and was not present at either school during the process. BSU is the smaller of the two schools and commonly held one or two sections of both CS0 and CS1 each semester. The instructor for CS1 differed for each term in which measurements were taken. The CS0 instructor remained a constant, although it was possible that some members of the control group had attended the CS0 course in a much earlier term. MSU is a much

larger school and runs multiple sections of both courses each term. The CS0 instructor remained constant while the CS1 instructors differed. The CS1 sections represented a subset of all CS1 sections in the school, since the sample only contained students in sections with faculty members who had agreed to participate in this study.

Data collected from the BSU campus was sufficiently clean to allow paired analysis of collected data points, so information from this school was used in the data analysis process. The MSU data was not analyzed, since data collected for the control group could not be successfully paired due to the use of multiple student identifiers at the beginning and end of the course. Failure to obtain a control group for this school made the inclusion of paired data in the treatment group impossible, and precluded the use of the data obtained.

The research population consisted of students enrolled in pre-programming courses at BSU and MSU during the Spring 2000 and Fall 2000 semesters. Learning was measured for these students in the first programming course during the Fall 2000 and Spring 2001 semesters.

1.4.7 Motivation for this Study

The researcher believes that the integration of problem-solving techniques and algorithm development tools early in the learning of computer science enhances the acquisition, retention, and application of computer science concepts. Experiential knowledge, acquired as a computer science instructor, indicates that most students need to be given tools to solve problems, to design solutions, and to test these solutions. If taught early in the curriculum, these tools could be useful to students as they learn to program and participate in the development of large projects for their coursework.

1.5 Original Contribution

Measuring the effectiveness of content changes to the pre-programming course supplements the current body of knowledge in computer science education. There has been a long-standing dialogue among educators with respect to introductory course approaches. The incorporation of algorithm development tools and problem-solving techniques at such an early stage in a degree program is not an entirely new approach, but controlled studies in this area do not currently exist. Therefore, proponents of this approach have very little to support their case other than their own classroom experience. While such experience is valid in its own right, it is difficult to initiate the implementation of these ideas without data to substantiate the claims. In addition to providing some support for this specific modification in curricula, it is hoped that this study will inspire future research on approaches to teaching computer science to new students.

Many studies that exist on this topic--such as McCauley and Jackson (1999), Bouvier (2003), and Apple and Nelson (2002)--tend to be incidental studies that fall prey to many design problems. This research brings additional strength to the data by providing direct evaluation of student accomplishment through a design using data and analysis triangulation. Additionally, the evaluation of learning took measurements that were not subject to the bias and pressures of course grades. For example, the instructor of record did not rate answers, so there were no additional data points with respect to a given student that might have affected assessment. Triangulation was employed to provide a more concrete and independent indication of curricular impact on learning for the individual. Most studies in computer science education use the results of existing graded events and existing curriculum materials. This research differs in that the data collection tools were not part of the course and the

'treatment' to the course came from a source outside of the organization that used it.

Computing professionals are likely to agree that the abilities to problem solve, pattern match, and develop structured solutions are vitally important in the field of computer science. The development of algorithms is relevant in hardware design, software design, database design, and most other computing related development fields. This study provides a sound design for research that can support the experiential knowledge of computer science educators. Discovering a correlation between knowledge of problem-solving skills and algorithm development tools, and the ability to effectively use a programming language, indicates that learning these concepts early in the process is beneficial to those looking to enter the field.

Perhaps the most important result that could come from this research is the continued acceptance of pre-programming curricula in computer science. The crowded computer science curriculum, combined with many students' desire to get directly to 'real computing,' creates pressure to jump directly to programming in the first course. In order to counter this pressure, effort must be made to gather evidence that supports the effectiveness of non-programming oriented materials in the introductory curriculum.

1.6 Social Relevance

It is the belief of this researcher that few computer science students are provided with an adequate foundation for software development in their computer science programs. This failure produces professionals who either require remedial training or who perpetuate a continuing cycle of misuse, abuse, or neglect in project development processes. While there are certainly other factors involved in producing such results, it is

advisable to critically examine computer science curricula and determine what components would improve the eventual success rate of new students. The development of new professionals who have an appreciation for problem-solving techniques and algorithm development processes, as well as an understanding of their relative strengths and weaknesses, should certainly serve to improve product development.

If computer science programs hope to develop new professionals who are comfortable with problem-solving and software or hardware development approaches, then an early introduction of this material will set an appropriate tone for learning. If students are given these tools early in the process, they can consistently refer back to them. Studies that investigate how computer science is most successfully taught will lead to the development of courses and curricula better suited to help new students enter the field. This will result in more adequately prepared professionals, and, in turn, better computing products.

In general, there has been very little research to support ongoing dialogue about how we teach computing and computer science topics. This study serves to supplement the body of knowledge about what is successful or unsuccessful in the early stages of learning this subject area. Hopefully, it will lead to additional research that may, in the long run, support more effective methods of teaching introductory computer science topics.

2 Intellectual Context

2.1 Computer Science, Problem-solving and Programming

Computer science is rooted in the problem-solving processes found in engineering (design), mathematical analysis (theory) and scientific experimentation (abstraction) (Computing Curricula 1991). It is, by comparison, a young field and, as such, it is rapidly changing. However, knowledge of the field's roots makes it possible to import ideas and concepts from these other areas, even if the specific body of knowledge is limited. For example, computer science shares many concepts and foundational theories with mathematics (Fujii, 1987). Therefore, it makes sense that computer science education research would benefit by importing research from mathematics education research (Almstrum, Hazzan, Ginat and Morley, 2002) and science (Almstrum, Hazzan, Ginat and Clement, 2003).

Problem-solving methods play a major role in computer science, as well as its parent disciplines. Strong foundations in domain recognition, pattern matching with prior solutions, problem organization techniques and solution testing are important for later success in the discipline. (Computing Curriculum 2001) Early collections of heuristics for mathematical problem-solving have relevance in current computer science problem-solving. In fact, one of the earliest sets of heuristics can be mapped to programming related problem-solving (Polya, 1957). Other mathematically oriented problem-solving work by Wickelgreen (1974) and Adams (1974) provide additional tools and insights that are applicable to computer science. One of the first transfers of problem-solving techniques to programming (Mitchell, 1984) is still applicable with today's languages and theories.

Recent work specific to computer science problem-solving correlated various types of problem-solving approaches with computer science problem domains. Shin, Jonassen and McGee (2003) determined that problem-solving for well structured versus ill structured problems require different skills. Both require domain understanding and justification skills, but ill structured problems require “meta-skills” such as planning and task decomposition. Larger systems design and analyses tend to be ill structured, so organizational system problem-solving techniques are useful in software engineering environments (Ackoff, 1987, Krantz, 1991).

Collaborative problem-solving techniques are also an important part of computer science problem-solving. Computer science is, by nature, a collaborative discipline, often requiring team development and work (Cook, 1997). This makes it even more important that practitioners be made aware of team problem-solving techniques (Powers, 2002). In fact, many techniques and processes found in mathematics, engineering, and scientific inquiry were built with the purpose of providing a tool to communicate problems, methods, and solutions. For programming, methods such as diagramming and pseudocode are used to provide some format for ‘program language free’ problem-solving. Therefore, it makes sense to promote these tools as part of a collaborative problem-solving toolset.

Unfortunately, the transfer of problem-solving techniques to application in programming is a difficult step (Woods, 1996). This emphasizes the importance of providing appropriate learning tools and frameworks from which new students in computer science can work. Assuming that pre-existing problem-solving aptitude is sufficient for programming success does a disservice to all but the most self-motivated students. Furthermore, those who use programming as the tool to introduce problem-solving techniques make the process more difficult by forcing

multiple learning stresses on the student (Woods, 1996). Providing intermediate steps between problem-solving techniques and programming support the concepts of an advanced organizer (Ausubel, 1968). The transfer of problem-solving skill to programming language implementation should be more successful if learning steps are clearly outlined for the student.

2.2 History of Curricular Change in Introductory Computer Science

Some of the first pre-programming courses appeared in the 1960s when flowcharting classes were required in many community colleges prior to the first Fortran programming course (Mitchell, 2001). Introductory courses tended to provide a breadth-based set of materials that provided a map to the field's domain. Part of this set of background information was the use of tools such as flowcharting and pseudocode (Mitchell, 2001). As colleges began adding computer science programs, it was natural that they should follow this template for their own curriculum, since there was little public knowledge about the field.

Certainly one can argue that much of the reason for this approach was the limited amount of computing resources found on post-secondary campuses. If a person had to create a set of punch cards for a simple program and pay for every minute of processing time, it made sense to use tools that were generally available. Thus, there were monetary and logistical motivations for pseudocode, diagramming and other modeling approaches, in addition to the pedagogical reasons cited here.

As computing power became more readily available, programming language learning was moved up in the curriculum. Nicholas Wirth's Pascal, which was developed as the 'learner's language,' provided a tool for the inclusion of a programming language as part of the early problem-solving skill set. A text by Leestma and Nyhoff (1984) provides a typical

example of a Pascal-based introductory course. The language was used as a problem-solving tool and other computing topics were built around the language. Because the language was designed with the student in mind, the difficulty of the learning curve was lessened somewhat, although it could be argued that it was still more difficult than pseudocode or flowcharts. Even so, one can see flowcharts or pseudocode used in Pascal texts with great frequency as a learning aid.

Movement away from Pascal to other languages (such as C, Modula-2, and Java) that were product oriented rather than learning oriented made the programming first model problematic (Wolz, 1997). The increased demand for learning and understanding syntax complexities pushed the focus on problem-solving and the rest of the computer science field into the background. Unfortunately, this fed the misperception that computer science was literally equivalent to programming. In response to these problems, the Computing Curricula recommendation for 1991 espoused a breadth-first approach, suggesting that the computer science field be approached initially in a survey-oriented course. (Computing Curriculum 1991). Various tools were developed--such as Karel the Robot (Pattis, 1994) and the Analytical Engine (Decker and Hirschfield, 1990)--in order to support the bread-first approach and training in problem-solving techniques that could transfer to programming.

It is also possible that movement to a breadth-first approach occurred, in part, because there was more interest in computer science by a broader base of students (Decker, 1992). In fact, there was an increased demand for departments to provide non-major courses in order to respond to the growing need for computer literacy (Curl and Hussin, 1993). Computers were rapidly being integrated into all areas of the college-wide curriculum and students were faced with the need to become proficient with these tools. There were numerous approaches suggested for the literacy program and their role in introductory computer science programs; for

example, one can review Goldweber, Barr and Leskal (1994). However, the call for computing literacy has moved into earlier learning curriculum and students now enter college programs with sufficient computing skills to make these courses less critical to the curriculum. The emphasis has returned to how the introductory sequence can best serve those who are entering the computing disciplines.

The breadth-first approach was not a complete success, nor did all computer science programs follow the Computing Curriculum recommendation. Some educators even began to worry that the discipline had become watered down and that the math background of the discipline was no longer valued (Tucker, Keleman and Bruce, 2001). Clearly, there was no accepted best practice for introductory courses in computer science. The net result was a new curricular outline that no longer supported a single approach to introductory computer science (Computing Curriculum, 2001). Instead, several curricular options are now supported by these guidelines, and computer science programs are left to choose their own best fit.

2.3 Introduction to Computer Science

There is no accepted best practice for the curriculum used as an entry point to the discipline. Instead, there are three introductory approaches supported in the recommended curriculum load for computer science. Essentially, these may be categorized as bread-first, depth-first and breadth-also approaches (Computing Curriculum 2001). This standard includes variations on these approaches, such as whether or not programming occurs in a breadth-first introduction and whether introductory problem-solving skills are addressed in a depth-first programming course. The breadth-also approach integrates both traditional approaches (breadth-first and depth-first), and encompasses multiple courses in a typical program.

Breadth-first approaches are supported by texts such as Brookshear (2000) and Schneider and Gersting (1999). Both texts rely at various points on programming languages to cover some of their content. However, they both limit this exposure and neither uses the programming language as the focal point for the entire text. It has been suggested that the best solution for a breadth-first introduction is to consider it an introduction or orientation to the subject area and to the department. Cook (1997) suggested that problem-solving skills should be given the largest portion (40%) of the course time, but other items (such as teamwork, computing tools, career options, subject options, and school/department related information) should be included. Cook also supported the option of omitting programming from the subject domain, although he did not hold firm to making this course a first course in the program. Others who have outlined breadth-first programs include those who are more oriented to discrete structures (Tucker, Barker, Bernat, *et al.*, 1998) (Tucker and Garnick, 1991) and those who tend to support a software engineering based approach (Bagert, Marciy and Callani, 1995).

The depth-first approach is commonly a programming first approach, although some incorporate problem-solving, algorithms, pseudocode and diagramming. Some use an alternative focus, such as web design, to attract interest (Mercuri, Herrman, and Popyack, 1998). However, the language is almost always the focal point for problem-solving skill development. Other proponents for the depth first approach go so far as to argue that programming should occur first and be followed by a breadth-based course (Gray and Frazier, 2002). In this case, the advocates were looking at retention and cite mathematical shortcomings as a barrier. Of course, this fuels the argument that the curriculum is in danger of becoming over-simplified (Tucker, Keleman, and Bruce, 2001),

as the justification for this change is obviously based on the preparation level of incoming students.

The current, most popular approach for those writing on the topic seems to be the 'breadth-also' approach, which seeks to combine both programming and breadth in the curriculum. This seems like an attempt to get back to the old environment often found in the Pascal courses, where a language built for learning eased students into organized/programmatic problem-solving. In order to achieve a similar environment, some espouse the use of 'simpler languages' such as Javascript (Reed, 2001).

A carefully presented and justified breadth-first approach is outlined by Powers (2002). This curriculum treats the first three courses as an introductory sequence. It is based on the concepts of spiraling (referring to topic areas multiple times at different levels of difficulty and different perspectives throughout the curriculum), spacing (the view that students need to see things over time in order to retain learning), and constructivist theories (the view that people construct their own understanding from what they have experienced). This approach is supported by the Computing Curriculum (2001) guidelines and is carefully backed by pedagogically sound choices. However, it remains to be seen whether these choices lead to increased success for participants in the program.

Numerous specific approaches to introductory classes are being tried with varied success and with varied attention to measuring that success. Some of these approaches focus on providing specific events, others argue for particular pedagogical frameworks, and others focus carefully on only one aspect of teaching in an introductory course. On the positive side, computer science educators appear to be quite willing to share new ideas and approaches to teaching with each other. On the other hand, measurement of success is usually based on typical student satisfaction

feedback, anecdotal evidence, and (possibly) grade results. Furthermore, results are generally specific to the school or program in which the results were collected, so making generalizations based upon them is usually quite difficult. Although there are shortcomings in the current collected data set, it is still pertinent to the body of knowledge for computer science education.

Common learning experiences that use active learning techniques to imprint a concept in the student's mind can be used in breadth-first environments (Lewandoski and Morehead, 1998) and in the depth-first curriculum (Bouvier, 2003). In both cases, educators used physical models and active events to encourage attempts to solve fun problems. For example, Bouvier used an exercise where students form a living flowchart, assigning roles to individuals for variables and processes. The intent of these exercises was to provide a base for understanding that students can easily relate to and to make the experience highly memorable for increased retention of knowledge.

Language selection has long been a hotly contested component of the introductory computer science curriculum. In fact, the language choice has been known to impact the entire computer science curriculum in a program (Dingle and Zander, 2001). Obviously, language choice matters most in a programming-first environment, since it is intended to be both a tool for problem-solving and for learning programming concepts. Pedagogical choices must also be weighed against student perceptions of viability and name-recognition. Some have attempted to formally analyze which languages make the most sense in an introductory class: For example, Isaacson and Scott (2002) tried to differentiate between Python and TCL, with minimal success. However, such studies have used highly subjective measurement approaches and have provided inconclusive results.

Regardless of language, another debate has arisen with respect to the use of graphical environments and built-in libraries. Some, such as Koffman and Wolz (1999), encourage avoidance of these tools, while others advocate for the full use of GUIs and built-in libraries (Odekirk, Jones, and Jensen, 2000). Of course, as with any polarized environment, there is always the possibility of attempting to hybridize the two, which is espoused by others (Comer and Roggio, 2002). Certainly, graphics grab student attention and may increase student satisfaction, based on surface reactions. Visualization projects have become a hot area for computer science education research and have crept into the introductory sequence, although the focus has been primarily on data structures and algorithms at later points in the curriculum (Akingbode, Finley, Jackson, *et al.*, 2003). An excellent set of criteria for effective visualization tools has been outlined by Naps, Rosling, Almstrum, *et al.* (2002). Anyone who plans to use visualization in an introductory course should look carefully at the recommendations outlined in that paper before proceeding. The authors make it very clear that surface satisfaction does not necessarily lead to better learning and that there must be a carefully considered strategy for including visualization as a key component in the curriculum. Visualization and creativity for problem-solving learning may have been best represented by Karel the Robot in earlier introductory courses (Pattis, 1981). However, some are now expanding the concept to using inexpensive robots as a problem-solving learning tool (Schumacher, Welch and Raymond, 2001) or as a programming learning tool (Schep and McNulty, 2002). These tools are intended to encourage students to work on problem-solving and programming more often by making the results very obvious to the student. It is supposed that students can better view consequences of algorithmic choices, and thus relate those choices more easily to outcomes, when visual and (potentially) tactile or aural senses are included as output.

Others approaches in introductory courses use tools that provide instant feedback for learning. For example, self-examination or self-testing tools increase the immediacy of feedback to the student. One such project is outlined briefly by Wabel (2000), where a self-quiz tool was built for students in Javascript. Another way of accomplishing interactive, but non-personal, instruction is the development of online tutorials (Ericson and Rogers, 1996). Usually, such tools are provided at schools with a higher ratio of students to instructors. Since it is more difficult for a student to receive individual attention, it is argued that such tools can serve as an adequate proxy.

Yet another approach is to link software engineering principles with the introductory sequence. In this case, it is argued that the software development process embodies a structured problem-solving approach that can be applied to large portions of computer science (Long, Wedie, Bucci, *et al.*, 1999). Hilburn (1993) outlined a highly structured curricular approach that directly reflects structured software engineering processes. This would be an instance where the entire curriculum is nested in software engineering principles. Some directly integrate software engineering into the laboratory environment only (Roberge and Suriano, 1994). But most proposed integrations, such as Towhidnejad and Salimi (1996), fall into the trap that assumes computer science is largely a software engineering degree. This is not necessarily an issue if one prefers the depth-first approach since concentration is already focused on learning programming. In that case, it makes sense to encourage correct development of software from the beginning. However, breadth-first proponents may not agree that this approach is fully extensible to all of computer science.

Specific works on methods for teaching programming include techniques that encourage programming from 'scratch' versus those that use a 'completion strategy' (Chang, Chiao, Chen and Hsiao, 1997). Educators who believe in the completion strategy tend to agree with Deimel and Moffat's (1982) outline of a successful programming-learning model. This model includes the observation of program execution and the study of good programs, followed by the chance to extend and modify those programs. Once students have had a chance to observe and work on a sufficient number of 'good' models, they can attempt independent design and code writing.

Another excellent idea for encouraging success in both programming and problem-solving skills is to connect assignments into bigger, more meaningful projects (Yang and Wei, 1999). Early courses often isolate processes and concepts into small tasks without providing a reference point within a larger, more comprehensive framework. An interesting study by Carbone, Hurst, Mitchell and Gunstone (2001) clearly indicated that students frequently fail to associate earlier tools and tasks with current tasks or tools. Projects that provide a broader frame of reference should provide linkage between concepts and tools and avoid the perception that each course task is isolated and unrelated to other tasks.

Another interesting approach incorporates reflection into introductory computers science activities (Fekete, Kay, Kingston and Wimalaratne, 2000). Part of the process of learning is integration, which comes from reflection about what has happened during learning events. In this case, the authors attempted to provide a tool to encourage useful reflection on learning in the introductory course. Tools, such as learning journals, are frequently used in other disciplines with success, and it makes sense to extend their use to early computer science courses.

2.4 Related Computer Science Education Research

Computer Science Education is still a very young discipline and research is still sparse, although growing rapidly. Unfortunately, a large number of articles purporting to be 'studies' or 'research' appear to be anecdotal or coincidental. It has been suggested that once some sort of criteria for research is agreed upon, the body of knowledge will become more stable (Clancy, Stasko, Gudzial, *et al.*, 2001). Others suggest that the body of knowledge should be supplemented by the integration of mathematical education research (Almstrum, Hazzan, Ginat and Marley, 2002) and science education research (Almstrum, Hazzan, Ginat and Clement, 2003) in order to build up this body of knowledge and inform new research.

Much current research is focused on determining predictors for success in various learning tasks. One of the earliest of these studies attempted for computer science failed to link high school test scores and performance with predicting success in computer science at the college level (Butcher and Muth, 1985). Another approach to finding predictors is to determine if there is a link between the successes in one course with success in another. Stein (2002) found some correlation between success in the first computer science course and success in the second course, as well as a link with the first calculus course and the second computer science course. Surprisingly, no link with a discrete structures mathematics course was found. This is counterintuitive, since discrete structures are viewed as a key component for computer science theory.

It is possible that those who enter introductory computer science courses with prior programming knowledge may have a decided advantage over those who do not. Hagan and Markham (2000) found that there was such an advantage and that it grew significantly if there was exposure to multiple languages. However, it should be noted that this approach assumed a programming-first curriculum, and success would naturally be

more likely in that situation. Furthermore, the extent of prior programming knowledge would certainly have an impact on learning. Persons who have limited introductory knowledge or who are misinformed about the process may have greater difficulties than those who have no prior experience.

Student characteristics may also provide insight into achievement in programming and computer science. In general, personality traits have been shown to have an impact on achievement (Diseth, 2003). Characteristics such as openness, conscientiousness, and neuroticism were shown to have some correlation with achievement levels. Deep, surface, and strategic learning approaches were also shown to have some effect on success in learning. This information is not necessarily surprising; it makes sense to say that a person who is conscientious usually achieves at a higher level. Similarly, a person with only a surface learning strategy will tend to achieve at a lower level.

The concept of self-efficacy is that individuals develop perceptions that can impact their ability to achieve. Bandura (1986) clearly outlines factors that may change or mold a person's individual perceptions (prior accomplishment, observed learning, emotions, and persuasion). These perceptions can alter overall performance, in part because the individual's choice to pursue or continue with studies are strong or weak. Some studies, such as Black and Deci (2000), show that those who have a more specific reason for undertaking a task or learning goal will tend to succeed. Therefore, one might extrapolate that a person who views becoming a good programmer as a key component of success in computer science will tend to do what is needed to succeed. Similarly, one could then argue that educators could alter perceptions about what is important by working on this desire to succeed and by linking problem-solving techniques to that success.

Overall performance could also be altered by the amount of effort and persistence put forth by the individual. According to Bandura, effort is influenced by the individual's perceptions of the task. This is confirmed by a study that found that attitude towards failure had a significant impact on achievement, whereas the process of learning appeared to have little impact (Bay and Daniel, 2003). Also, research specific to CS1 by Rountree, Rountree and Robins (2002) found that the strongest indicator for achievement (of several possible indicators) was the expectation the individual had for success. These contentions might lead one to extend the results and conclude that there is no need to worry about approach in introductory courses. After all, process doesn't matter. However, it is important to remember that attitude can be reformed and placed into a new context (Bandura, 1986). Therefore, attitude can be altered by the approach, techniques, and coverage used in introductory courses, when foundational contexts are being built.

Apple and Nelson (2002) identified several risk factors for introductory computer science students and outlined approaches they intended to take in order to alter the situation. Many of those changes focused on attempts to change attitudes in hopes of encouraging success in learning. Changing attitudes and combating myths about problem-solving are both goals for the introductory computer science sequence. Efforts to isolate beliefs about problem-solving learning have shown, for example, that new students tend to believe there is more value in speed tests (cover lots of short problems) versus power tests (fewer but deeper problems) (August, Lopez, Yokomoto and Buchanan, 2002). Similarly, they tended not to do additional problems beyond assigned work. Since these are known to be poor choices in problem-solving learning, these attitudes and beliefs could be addressed in an effort to change rates of achievement.

Learning styles can also be linked to performance in early computer science courses. Thomas, Ratcliffe, Woodbury and Jarman (2002) found that students who could be categorized as active, sensing, or visual students tended to do poorly. To their credit, the researchers have not attributed the success and failure of these students as natural tendencies that lead to failure in computer science. Instead, they have correctly assumed that it is necessary to find ways to make the curriculum more inclusive of varied learning approaches. This is consistent with Markova (1996) and the 'six patterns of natural intelligence,' where it is clear that tendency need not dictate success. Furthermore, the authors used results in mathematics education research by Tanner and Jones (1999), which shows that teaching style has an impact on thinking as it either succeeds or fails to facilitate for certain learning styles.

Characteristics more commonly linked to success in computer science include comfort with mathematics, computing and problem-solving. Byrne and Lyons (2001) show a tendency for persons with prior mathematical and science skill ability to have an easier time with learning programming. Gould and Rimmer (2000) followed cohorts of students through the introductory classes and found that problem-solving skills are important for programming success. Furthermore, a dislike of programming was an indicator for success or failure in the introductory sequence. As with many of these studies, it is difficult to determine whether this is a product of how courses are currently taught, or whether it is a determining factor in predicting success. However, one could argue that this research indicates that a programming-first approach could be less desirable.

Other studies attempt to build self-efficacy measuring sticks for varying classes, tasks, and problems, and this seems to be a more reliable method for obtaining sets of predictors for success. For example, a self-efficacy scale was created for programming success (Ramalinam and

Wiedenbeck, 1998) that appears to have reasonable reliability and validity measurements. Similarly, an excellent study by Quade (2003) developed a scale for CS0 courses and found that prior problem-solving success, along with other characteristics, tended to play a role in CS0 course success.

Studies specific to approaches taken in introductory computer science education tend to be used to measure levels of success for favored approaches. For example, Bouvier (2003) used data collected with regard to student satisfaction, retention, and grades to determine the effectiveness of common learning experiences in the depth-first introductory environment. While this sort of study is not well controlled and is subject to numerous external variables, it is interesting to note that retention and satisfaction increased while the grade distribution did not improve. Lewandowski and Morehead (1998) used a similar study approach and reported positive impacts on learning in their breadth-first environment.

McCauley and Jackson (1999) attempted to measure the success of integrating software engineering into the introductory curriculum. In this case, the researchers tried to determine if this change affected future learning by tracking students through the computer science program and collecting grades for subsequent classes. Again, the subjective nature of the data collected and the numerous external variables were bound to counter the modest improvements found in certain courses. On the other hand, Buck and Stucki (2000) argued, without the benefit of a study, that Bloom's taxonomy prohibited success of the integration of software engineering. They may certainly have a point, given the fact that some attempts at this integration do not provide a proper framework from which students can construct knowledge. However, this observation should not

be extended to every attempt to incorporate software engineering into the introductory curriculum.

Perhaps one of the most specific and innovative pieces of research is that undertaken by Booth (1997), who used phenomenographic research methods to understand how students learned recursion. The results of that study provide interesting insight on methods for presenting that topic, but its extensibility to the entire introductory sequence is problematic. However, those who use Scheme-language approaches as an introduction to computer science will find this work useful.

Pedagogical studies that have direct relevance for introductory computer science strategies include one by Dempster (1988) that clearly indicates the benefits of spacing in the curriculum. In other words, it is important to space a concept or specific learning task/area over a prolonged period of time to increase retention of knowledge. This study strikes a blow to curricular structures that over-compartmentalize topics. In particular, any introductory sequence that neglects repetition or linkage of concepts to the rest of the curriculum is bound to see lower success rates. Hazzan (1999) argues that abstraction levels in algebra need to be reduced in order to reach beginning students. This study could easily be applied to early problem-solving in computer science, since it supports techniques that incorporate some sort of concrete method to emphasize concepts. A more general study by Davy and Jenkins (1999) attempted to measure the difference in programming learning when the course implemented components based on pedagogical models that had seen general success. The new course consisted of a discursive component (learning goals agreed upon and clarified), an interactive component (collaboration with peers and instructors), an adaptive component (to respond to change and individual situations), and a reflective component. The results of the

study, although inconclusive, did not show a decline in learning achievement.

Current studies in paired programming are finding that collaborative work may not harm, and possibly may help, students in beginning programming courses. A study by McDowell, Werner, Bullock and Fernald (2002) shows that students completed the first programming course at higher rates and performed nearly as well on individual exams as those who programmed individually. While this is not conclusive, it does lend further support for collaboration in the early computer science curriculum. Unfortunately, collaboration is also a learned technique and Johnson, Johnson and Smith (1998) clearly address the need for instruction in teamwork and collaboration, if one expects to use it as a pedagogical tool for learning.

A strong body of knowledge exists regarding the need of students to understand the goals of any given course of study. Many attempts at implementing a computer science curriculum have produced students who did not know the reason they were in the program, even after several semesters of attendance (Howell, 1996). This is, perhaps, indicative of a bigger problem. Students are entering computer science with inaccurate preconceptions of what the discipline's foci are. Many believe the myth that programming is computer science, and vice versa (Powers and Powers, 2001). If one agrees with the constructivist philosophy that students build their own understanding by combining prior understanding with new experience, it makes sense that an introductory curriculum should work to quickly establish a framework upon which future learning can be built. Ben-Ari (1998) clearly established that this lack of framework leads to future learning problems in mathematics; he then extended this work to computing and programming (Ben-Ari, 2001).

Research also shows that students need to be encouraged to try problem-solving approaches that do not naturally occur to them. For example, it has been shown that new students tend to spend very little time analyzing a problem and rely almost exclusively on trial and error (Schoenfeld, 1992). This approach is quite contrary to that taken by experts, who use a combination of associations, intuitions, and testing to solve a problem (Fischbern, 1987). It is clear that less experienced problem-solvers could use direction and encouragement to begin using a broader set of tools.

Finally, a large body of knowledge exists that clearly shows that working from within some sort of advanced structure tends to aid students who enter a new domain of knowledge. Research in this area can be traced back to Dewey (1916), who believed that learning was a series of reconstructions of knowledge. Ausubel (1968) applied these ideas to language learning; he found that, to learn a language, one first perceives how the language works or applies, then subsumes that knowledge using experience as a frame of reference. Only then can one apply language learning. Mayer (1981) found that, in general, advanced organizers (tools that help provide a frame of reference for a new student) did improve learning success. Perfetti (1979) confirmed that advanced organizers tended to increase retention. Holt, Boehm-Davis and Schults (1987) extended this to programming by showing that object oriented or functionally decomposed programming solutions were understood better by new programmers than less organized code. In other words, programming that had structure was more successfully learned. Work by Bailie (1991) illustrated that existing coding models supported programming learning. On the other hand, Frazer (1998) demonstrated that existing models seemed to have no impact on those learning a second or third programming language. Finally, Applin (2001) specifically showed that those who learned to program by starting with programming

templates performed better than those who learned with no starting template.

2.5 Context within Body of Knowledge

This study, by necessity, worked within the breadth-first, CS0 approach to introductory computer science. Both schools who agreed to participate work with a curriculum that uses this technique. This does not preclude the increased use of problem-solving and algorithm development techniques in depth-first or breadth-also curricula. This research focused on the addition of a higher concentration of materials in this area prior to programming and was only tested within the bread-first environment. However, it is possible that the results of this study could provide some insight to alternative curricular structures in addition to the breadth-first approach.

It must be stressed that the approach taken in this research project assumed that problem-solving, pseudocode, diagramming, and algorithm testing would be covered prior to any coverage of programming. No specific significance was attributed to other topics presented in the CS0 course, since they were the same for both the control and treatment groups. Furthermore, no claims were being made with respect to the relative value of any of the major strategies (breadth-first, depth-first, and breadth-also).

Current pedagogical knowledge and recent study results were utilized to identify an appropriate set of curricular changes and their corresponding measurements of programming learning change. First, the curriculum achieved both spacing and spiraling for students in the program. This was accomplished by covering problem-solving within the context of general problem-solving, then within the context of diagramming, pseudocode, and testing. It was revisited in the CS1 course as a part of programming

learning. Furthermore, by presenting problem-solving in a general sense early in the first course, more students were better able to construct a new knowledge of problem-solving, closer to what they would need for programming. Introducing diagramming, pseudocode, and algorithm testing provided the intermediate step between general problem-solving techniques and programming. This served as the advanced framework into which programming in any language could be placed.

The alterations to the curriculum did not ignore the issue of student satisfaction. Instead, the new curriculum provided a clearer framework and purpose for the CS0 course. Better definition of purpose allowed students to prepare themselves for success and provided them with a link from current tasks (algorithm development) and future tasks (programming in a language). Also, this technique isolated learning curves and reduced the stress of reacculturating to a new body of understanding and knowledge (Bruffee, 1993). It was expected that a better focus for the CS0 course would lead to improvement in future success rates in the program.

Admittedly, part of the purpose for providing improved algorithm development tools in the CS0 course was to level the playing field for persons who either have had less prior problem-solving success or for those who have had no prior programming experience. Persons who have already completed programming tasks successfully are, in fact, likely to succeed again. Although they could certainly come away with some new understandings and new tools, such outcomes were not the focus of these curricular changes. On the other hand, it was expected that persons with prior programming experience might find these tools to be useful, since they would be able to more readily generalize the process and move from one language to another. It was also deemed possible

that these people might better learn certain concepts after exposure to this new curriculum.

3 Methods

3.1 Research Design: Study Structure

3.1.1 Research Question/Goal Statement

Will the integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course impact the learning of programming skills and application of problem-solving skills in the first programming course for post-secondary institution students?

Algorithm development concepts and problem-solving techniques were delivered to participants by integrating new materials with the existing CS0 courses. This does not imply that there was an absence of problem-solving or algorithm development techniques in the CS0 course prior to the treatment. If that had been the case, it is likely that the results would be more definitive. Instead, changes were carefully integrated into an existing curriculum. The new materials expanded coverage on problem-solving and algorithm topics and increased their relative importance for the course. Details of these changes can be found in section 3.3.2 and all added materials may be viewed in Appendix I. The new materials were designed specifically for the breadth-first approach used in the existing courses. However, this does not preclude the validity of these results for general inclusion of these topics among the diverse options for CS0 curricula.

Problem-solving techniques encompass the identification and understanding of a problem as well as the discovery, testing, and communication of possible solutions to that problem. The process begins with requirements gathering, where vital facts are identified, the scope of the problem is confirmed, background information is discovered, and the goal is clarified. Possible solutions are explored using various techniques, including the development of algorithmic solutions for the problem. These solutions must then be tested for correctness and completeness.

Communication and collaboration are also very important parts of the problem-solving cycle. The modified curriculum for the treatment group provided an introductory-level coverage of these basic tools.

Algorithm development concepts focus on the creation of well-defined, step-by-step solutions for a given problem. Computer science solutions tend to focus on process or procedure, so an algorithmic solution is frequently chosen for problem solution communication. Algorithms are represented in pseudocode, diagrams, program code, and/or mathematical terms. There is a broad range of programming languages that are used for the implementation of algorithms. Because these languages bring with them specific syntax and structure requirements, it is useful to have these generalized methods of representing algorithms. Since mathematical representations, aside from proofs, are not flexible enough to represent all process-oriented solutions, pseudocode and diagrams are most frequently used as solution communication tools. Once the solution is proposed in a standardized and generalized form, the algorithm may be tested for correctness and completeness.

In practice, algorithm development approaches are recognized as useful tools for programming and project development. In fact, programming is simply an application of problem-solving, using a specific grammar to enact proposed solutions. Intuitively, there is a direct mapping of algorithm development skills to programming skills. It is logical to identify these topics as appropriate advanced organizers for programming learning, so the new CS0 curriculum provided students with a basic set of algorithm development and testing tools for use as tools for the representation and communication of candidate solutions.

There are, of course, several types of programming languages. While this study makes no distinction between these languages, the subject courses

used the C++ and Java programming languages. Both languages include object oriented programming, and both have a basis in traditional procedural programming structures. Therefore, programmers for each of these languages may benefit from learning step-by-step algorithmic approaches. Languages based on lambda calculus (such as Scheme or Lisp) may not benefit as strongly, since they are built for recursive solutions. This study did not include lambda calculus-based languages in its scope because no tools were incorporated to clarify recursive concepts in the curriculum. However, it should be noted that algorithms could be presented in such a fashion that recursive solutions would be supported, if the goal programming language required it.

The research question limited the research domain to two courses in traditional post-secondary computer science courses. Thus, educational efforts in primary and secondary schools may not be able to apply the results of this research, since the population and environment differs significantly. The target population consisted of adult students enrolled in typical four-year degree programs in the United States. This limited scope was largely driven by the convenience of willing and available instructors of CS0 and CS1 courses at two institutions fitting this description. However, these limitations do not preclude the possibility of future adaptation and study in other populations or dissimilar environments.

3.1.2 Study Type

The intent of this research was to provide evidence that a hypothesis or its corresponding null hypothesis should be accepted or rejected, which indicated that the research was explanatory in nature. However, there was also a descriptive research component, since the breadth of the data collected included information that did not directly test the hypothesis. This information was used to account for some of the external variables that could not be handled by using randomization techniques. Therefore,

this project was exploratory research that combined explanatory and descriptive goals.

This was a control study where groups existed naturally as cohorts. Individuals were not selected and assigned to groups, so a nonequivalent control group design was used. The research design modified the 'cohort design with treatment partitioning' described by Cook and Campbell (1979). The biggest difference between this study and most pretest/posttest control group designs was that the treatment actually occurred before the pretest. It is more accurate to say that this study consisted of a posttest in two time intervals; however, "posttest in two time interval" models assume that the treatment pertains directly to the object of measurement. In this case, the goal was to measure subsequent learning rather than the learning of the materials provided in the "treatment" curriculum. Since those who took the CS0 course prior to modification had some exposure to algorithms and problem-solving, the control group was more accurately labeled as a "weak" treatment group. Participants who took the course after modification were part of the "powerful" treatment group. The pretest became, in effect, a first interval, post-treatment measurement, but will be referenced in this document as the pretest for simplicity. The posttest was a second interval, post-treatment measurement and represented the first opportunity to measure programming skill and knowledge.

3.1.3 Triangulation

Triangulation occurred at both the data collection and data analysis points of the study design in this research. This implies that there was triangulation of the purpose or goal of the project. Although the research question remained the overarching goal of the entire piece of research, subordinate hypotheses were used to lend greater clarity to the primary question. A standard approach for research would have been to develop

a hypothesis and a null hypothesis for which data would be collected in order to support their acceptance or rejection. However, the weight of external variables made it unlikely that truly useful conclusions could be reached with this approach alone. Therefore, the research question was used to develop complementary data collection tools that would integrate with the standard hypothesis testing approach.

One part of triangulation was the creation of subordinate research questions, developed in response to findings in the literature review. Participant satisfaction and understanding of the relevance of covered material (see Chapter 2) has been shown to prepare students for successful learning, so questions were included in the data collection tools that gauged student satisfaction with the pre-programming (CS0) course as it related to the CS1 course. The rationale was that if students showed a significant increase or decrease in satisfaction in the CS0 course, this information would provide or remove support for the overall hypothesis.

A second subordinate research question was built around student self-evaluation of skill. Existing research indicated that students tend to learn and perform better when they feel that they will be able to succeed (see Chapter 2). Self-evaluation questions were included in the data collection tools and these data were applied towards testing a second subordinate hypothesis about student self-evaluation of capability. If the data revealed increased or decreased confidence in ability to perform programming tasks, for example, then it increased or decreased support for the overall hypothesis that curricular changes positively impacted learning.

In addition to subordinate research questions and hypotheses, qualitative information was also gathered. Some of this information was collected by means of open-ended questions in the demographic data collection tool and the exit survey data collection tool. Additional qualitative data was

collected for approaches used to solve problems in the pretest and the posttest. For example, it was noted whether participants used diagrams, pseudocode, or other tools to help solve problems. Collection of these data points stemmed from an interest in exploring the research question, but data collected in this fashion did not lend itself to hypothesis testing. Qualitative analysis was utilized to confirm or clarify the findings of the hypothesis testing methods.

The final piece of the triangulation process in analysis was the addition of data mining on information collected by the data collection tools. The volume of external variables made it highly unlikely that hypothesis testing alone could make a strong statement about the changes enacted in the curriculum. Unsupervised clustering was used on instances of data, using the participant identification number to differentiate between instances (records). External variables suspected to have impact on the results were included so that clusters driven by these pieces of data could be identified. When this process uncovered potential correlations for external variables, they were tested further for significant impact on the test score outcomes. Further data mining analysis used supervised learning to test specific, learning-based categories. Two-thirds of the instances were used to build a model and the remaining one-third of the instances was used to test the viability of that model. Data fields selected for inclusion in these sessions were directed by the unsupervised clustering results.

3.1.4 Hypothesis Testing Structure

It was natural to select a hypothesis that represented both the desired result and the result that best matched experiential and anecdotal results found in the literature review. Therefore, the selection of this one-tailed alternate focused on an increase in learning for those receiving instruction based on the altered curricular materials. However, since it was difficult to isolate the variables sufficiently in order to accept or reject this hypothesis

with any reasonable degree of certainty, the null hypothesis was tested for acceptance.

Null Hypothesis

The integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course will result in NO MEASURABLE DIFFERENCE in the ability to learn programming skills in a subsequent course for college and university students.

The null hypothesis posited that there would be no measurable difference in learning between the control and the treatment groups. Acceptance of the null hypothesis would imply that the curricular change neither aided nor obstructed the learning of programming skills in the CS1 course. This, of course, would be an acceptable outcome from a pedagogical standpoint, as it would mean that no harm was done to participating students. Acceptance of the null hypothesis in this particular study was not viewed as a potentially definitive result, since all participants who attended a CS0 course did have some exposure to problem-solving and algorithm development. Furthermore, acceptance of the null hypothesis was not considered to be an indication that algorithm development and problem-solving had no value for future computer science learning. It would mean only that no real impact on early programming skills was demonstrated. Rejection of the null hypothesis was viewed as potential evidence that there was some measurable difference between these two groups. Given the possibly large set of external variables, it was expected that this would likely represent the extent that any positive or negative correlation between the treatment and programming skill success could be claimed.

Measurement of the “ability to learn” could be a source of much disagreement if one were to become overly concerned with semantics. For the purposes of this research project, the ability to learn programming

was measured by the evidence of knowledge shown by participants in the pretest for problem-solving and the posttest for programming. It may have been more correct to predict that there would be a measurable difference in exhibited programming skill than to state that the ability to learn would increase. Furthermore, it is possible to argue that "ability" refers to one's predispositions for learning and doing. However, the researcher was persuaded that problem-solving and algorithm development skills enhance the ability to develop programming skills. Therefore, evidence of accomplishment (or exhibited knowledge) over a time interval was taken as evidence of learning. Since measurements were not taken on how participants learned problem-solving or algorithm development, the relationship between algorithm development and programming skills were highlighted. Hence, the ability to learn was measured by examining participants' knowledge status at two points in time after the treatment or control CS0 curriculum had been applied.

Two-Tailed Alternate Hypothesis

The integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course will result in a MEASURABLE CHANGE in the ability to learn programming skills in a subsequent course for college and university students.

Alternate Hypothesis (One-tailed):

The integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course will result in a measurable INCREASE in the ability to learn programming skills in a subsequent course for college and university students.

The alternative (one-tailed) hypothesis reflected the researcher's experience and expectations about the alterations to the curriculum in the pre-programming course. However, rejection of the null hypothesis was not viewed as equivalent to acceptance of the alternate. The logical conclusion if the null were rejected would be that there was a measurable

change in learning, but since it was entirely possible that the change in curriculum might have led to confusion, dissatisfaction, or misdirection that decreased learning, a two-tailed hypothesis was included as an additional possibility. Acceptance of this hypothesis was considered an indication that the curriculum had an impact on learning. It was deemed possible that this change would be either positive or negative in nature, and equally possible that it would be both positive and negative. Regardless, acceptance of the two-tailed alternate was viewed as support for the contention that curricular decisions in the CS0 course were of potential import for subsequent CS1 courses. (Of course, acceptance of this hypothesis could not occur if the null was accepted. However, if this hypothesis was accepted, it would then be possible to accept the original, one-tailed alternate.)

Acceptance of the one-tailed alternate was considered to be the strongest statement that could possibly be made as a result of the data collected in this research. Since there was a distinct possibility that results would fall within the standard margin of error, triangulation was used to support the hypothesis-testing component. Consistent results across all methods utilized in this study were expected to help address issues of validity and reliability that could not otherwise be adequately dealt with in a pure hypothesis-test design.

Subordinate Hypothesis 1:

The integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course will result in a measurable INCREASE in CS1 student satisfaction as to the applicability of the CS0 course.

Satisfaction measurements were a reasonable inclusion in a study that aimed to improve the curriculum for a given course. Participants were informed that the motivation for the study was to improve the prior course

in the curriculum, so they were prepared to answer questions pertaining to their assessment of the course. This information was supplemented by qualitative information solicited by open-ended questions in the same data collection tools, which became an important part of the triangulation process. In light of the studies that link satisfaction with learning, evidence of increased satisfaction was viewed as support for the primary hypothesis.

Subordinate Hypothesis 2:

The integration of a set of algorithm development concepts and problem-solving techniques into a pre-programming computer science course will result in a measurable INCREASE in self-evaluation ratings for programming skills.

Self-evaluation ratings explored participants' confidence in their ability to perform given tasks. Poor confidence levels usually manifest themselves in poor performance; just as high confidence levels often result in successful attempts (Bay and Daniel, 2003). This subordinate hypothesis was provided as part of the effort to triangulate and increase the strength of each part of the research. Consistent results between the primary and subordinate hypotheses were expected to enhance the strength of each of the three statements.

3.1.5 Qualitative Structure

The qualitative portion was originally contained in three units; however, only two of these units were successfully integrated into the overall project. Preliminary designs for this study required the collection of programming samples from participants. Logistically, collection of such samples proved to be problematic, as was the ability to ensure that both groups provided equivalent work products. Furthermore, these samples would have been part of a graded activity, which would have been inconsistent with the rest of the data points. The removal of this unit did

not impact the results of the other data in this study and the absence of this data actually encouraged consistency in the results.

A similar set of open-ended questions appeared in both the demographic data collection tool administered at the beginning of the CS1 course and in the exit survey administered at the end of that course. Participants were asked to describe what they thought the most useful and least useful portions of the CS0 class were. Measurement at the beginning and the end of the CS1 class was intended to provide a tool for determining how increased experience changed reactions to the preprogramming course. Responses to all open-ended questions were hand coded using an abbreviation-coding system. Analysis of this information was independent of student identifiers and was not a part of paired analysis. Particularly insightful responses and commonly recurring themes were noted as a part of the analysis process.

In both the pretest and the posttest, participants were asked to solve four different problems. The introductory boilerplate read to the students at the beginning of the test encouraged the display of any and all evidence of attempts and tools used to solve the problem. After each answer was evaluated for correctness and clarity scores, the researcher cited evidence of various types of problem-solving approaches or solution illustration methods. For example, many individuals used a simple text paragraph to communicate their solution for a given problem in the pretest. This was noted as qualitative study information for the given student for that particular problem. This data was maintained along with the student identification and was available for paired analysis. Results were also coded into common categories for the data-mining portion of the research project.

3.1.6 Data Mining Structure

The data mining process was not initially planned for this project. However, the large number of attributes collected by the data collection tools, combined with the number of external variables inherent to measurements of teaching and learning, made data mining an appropriate part of triangulating the analysis of data. Data mining results that supported categorizing learning success based on treatment or control group exposure were considered to be extremely valuable, since such a grouping would have occurred despite the inclusion of several external variables in the data. While it was viewed as unlikely that such a strong result would occur, it was considered possible that equally valuable information could be gleaned if few, or any, of the suspected external variables showed strong correlations to learning. This was expected to be an indication that exceptional circumstances did not override the hypothesis.

The data mining process was undertaken by creating instances keyed on the student identifier. These instances included data fields created to represent data points from each of the four data collection tools. An additional attribute was created to indicate if an instance was a part of the treatment or control group, and another attribute was created to hold the difference between pretest and posttest scores. Some data fields, such as age, were broken down into ranges so that data could be treated as categorical, rather than numeric/continuous in nature. Other data fields (such as those asking about student intentions to major or minor in the subject areas) were intended for institutional use, and these fields were removed from the data mining format. Once records were created for each participant, the data was mined using a trial version of the iData Analyzer supplied as a part of the Roiger & Geatz text (2003).

Initial attempts at data mining included all records for all participants who qualified to be a part of the control or treatment groups. Any other records were discarded from the file to be mined. Data mining sessions using unsupervised clustering were initially undertaken in an effort to see if the iData Analyzer might uncover categorizations that successfully grouped instances that were similar to the actual control versus treatment group breakdown. In order to do this, the data field that indicated which group a person belonged to was masked out and not considered part of the input data. Numerous (15) unsupervised clustering sessions were undertaken with different combinations of attributes used as the input data for categorization. Outstanding and representative data mining sessions were selected from this group for analysis. Redundant sessions were discarded, as were sessions that could not produce a reasonable (two to five) number of categories.

Once unsupervised clustering results had been reviewed, supervised learning analysis was performed to determine if a rule set could be developed that would successfully classify new instances into the correct category. Supervised clustering required the inclusion of the group attribute (whether the instance was a treatment or control group member) as the desired classification variable. Other attributes were selected to be the input variables for determining the rule set. Two-thirds of all instances were used to train the iData Analyzer and cause it to determine rules for membership in these two classes. The remaining one-third was tested to determine if the rule sets generated could then be accurately classified. The ratio of control group versus treatment group members was maintained in both the training run and the test run for supervised learning data mining sessions, so each group was given a similar chance for training and testing. Test results were then placed in a confusion matrix in order to determine how well the generated rule set for class determination worked.

As with unsupervised clustering, there were multiple supervised learning sessions performed on the data set. Input attributes were selected using various subsets of all attributes. These choices were directed by results observed during the unsupervised clustering sessions. Of the ten sessions, a subset of three sessions was selected for detailed discussion. Other data mining sessions either exhibited similar results, or extremely poor training and test phase results.

3.1.7 Study Timeline

CS0 Preprogramming – Control Group	Spring Term 2000
IRB Approvals	August 2000
Beginning of contact with Study Participants	September 2000
CS1 Programming – Control Group	Fall Term 2000
CS0 Preprogramming – Treatment Group	Fall Term 2000
CS1 Programming – Treatment Group	Spring Term 2001
Conclusion of contact with Study Participants	Early June 2001

Table 1: Study Timeline

3.2 Research Design: Sample

3.2.1 Target Population

The general target population consisted of those individuals looking to undertake four-year, post-secondary study in the computing sciences. Since it was not possible to take a sample from a population involving a large set of institutions, the sample target population was limited to those persons beginning studies in the computer sciences at two medium-sized universities in the Midwestern United States. The results of this study are most likely to apply to institutions with similar populations and computing programs. However, institutions with populations for which this sample is not fully representative may also find some value in these findings.

3.2.2 Sample Selection

Sample selection differed slightly for each of the two participating institutions. Neither participating school advertised a difference in the CS0 course to their respective student bodies. Members of the treatment group and the control group were unaware of any changes made to the pre-programming course at the point of enrollment. Further, no controls were placed on CS1 courses in response to the altered CS0 curriculum. This resulted in a sample of the population from each group that was not subject to self-selection issues. The resulting samples approximated a random sample, based solely on the fact that persons registering for courses had no foreknowledge of the study.

The BSU sample was a time-based cross section of the target population at that school. All pre-programming sections during the Fall 2000 term received the new curricular materials and applied them during that term. All programming sections during both terms were involved in the study, since all instructors of these courses were willing to participate. Thus, the sample at BSU was representative of the entire population and can be treated as a cohort for analysis purposes.

The MSU sample differed slightly because, as a larger school, more sections of each course were concurrently offered. All participants in the Fall 2000 CS0 course were exposed to the new curricular materials. However, only sections of the programming (CS1) courses with participating instructors became part of the sample. The sample was representative because it was extracted from a time-based cross section of students who had no foreknowledge of which CS1 sections were run by instructors participating in this research.

.

3.2.3 Expected Sample Characteristics

A strong majority of students in computer science programs are male and most are of traditional college age (18-22 years). The location (Midwest United States) increased the likelihood that participants would be of Caucasian/European descent. The rural location made it more likely that persons would be commuters, especially if they were from a non-traditional age group. This rural setting also reduced the likelihood that a diverse American minority group would have strong representation. Computing majors do tend to draw stronger minority populations than many disciplines; however, most of these persons are from outside of the United States (Taulbee, 2000). There was expected to be a slightly larger instance of Native American participants given local concentrations of this population (Bemidji State University, 2000).

Persons participating in these courses were deemed likely to have had success in problem-solving, computing, mathematics, or pattern matching at a prior point in their lives (Quade, 2003). It was also considered likely that members of the sample would have had a reasonably significant exposure to computers, and that few would be unable to perform simple to intermediate tasks on computing equipment. This is a contrast to populations of students only five to ten years in the past that had little to no exposure to computing environments (Goldweber, Barr and Leskal, 1994). It was considered possible that there would be a minority group consisting of persons who struggled with problem-solving, computing and mathematics; however, most individuals who fell into this category did not meet the criteria necessary to be a member in the control or treatment group. In fact, many of the persons who fit this description did not participate in a CS0 course.

Finally, it was expected that the majority of students would be in their first or second year of post-secondary education. However, the number of later year participants is presently higher due to the increasing instance of persons taking more than four years to graduate. Also, since more programs grant college credit to high school students, instances of persons reporting sophomore, junior and senior status despite the fact that it might be a first or second year at the college was expected to be higher. It was also anticipated that most of the students in this study were likely to have selected these courses with the intent to major or minor for a computing degree.

3.2.4 Classification of Subjects

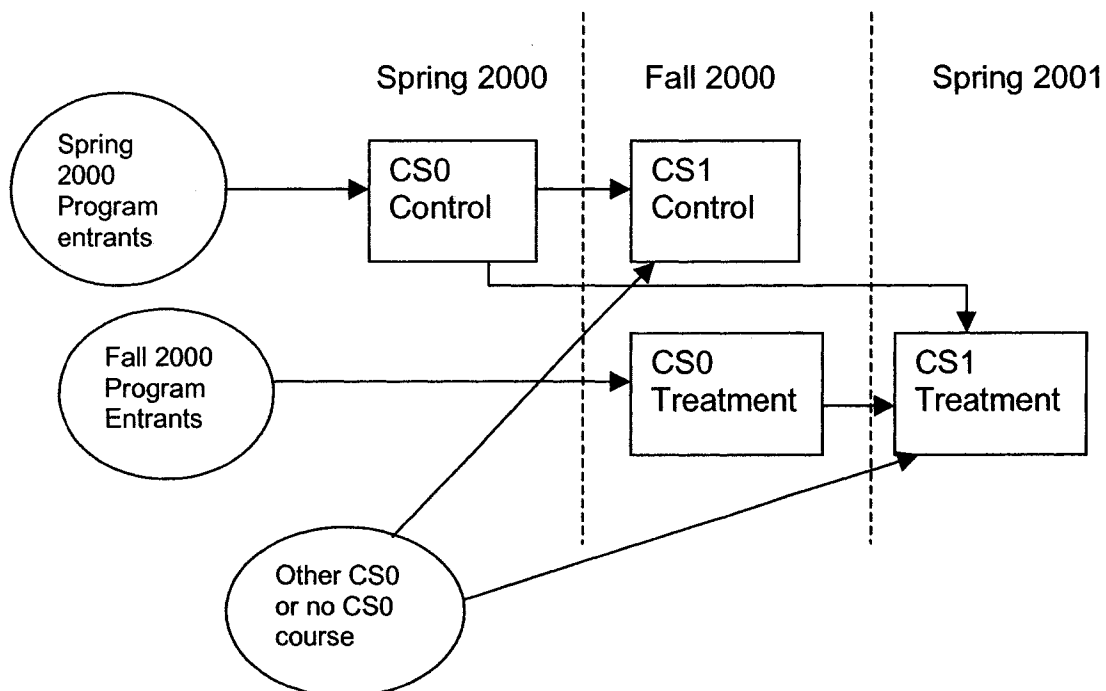


Figure 1: Control and Treatment Group Origin

Members of groups were not identified until enrollment in a participating CS1 course and the completion of the pretest and demographic data collection tools. These tools were implemented at the beginning of the

CS1 course and participants were given appropriate consent forms and an option to refuse participation without consequence for their studies in the programming (CS1) course. Since participants were not identified until the second course, it was important to identify possible sources for entrants into this course. Figure 1 shows a graphical representation of possible entry points and sources of participants in the CS1 programming course.

Candidates for membership in the control group were taken from all participants in the programming/CS1 course during the Fall 2000 term. No records were kept with respect to performance in the CS0 course, with the exception of self-reported grades in the demographic data collection tool. Individuals who failed to complete the CS0 course, or those who chose not to continue to CS1, were not part of the sample. Only persons in CS1 sections with participating instructors were part of this sample (all of BSU enrollees, selected MSU sections). The majority of persons in the control group were expected to have participated in the Spring 2000 CS0 course; however, no control was placed on enrollment to ensure that this was the case. Thus, it was possible that a small minority of members of the control group might have taken an earlier iteration of the CS0 course. Of this minority, it was most likely that these persons would have taken the course in the Fall 1999 term. The pre-programming course was not appreciably different at that time, so these individuals were included in the control group. Persons who received an exception to attend the programming course without completing the CS0 course were omitted from the study.

The term of completion for the CS0 course (or the lack of a CS0 course) was a data field collected by the demographics data collection tool. This provided information that allowed the researcher to separate instances into control, treatment, and outlier groups.

The treatment group consisted of persons who attended the CS1 course during the Spring 2001 term. As with the control group, the majority of the participants in these classes had attended the prior term's CS0 course. However, it was possible for persons who took another version of the pre-programming curriculum (or no CS0 course) to attend the same course as those who were part of the treatment group. The demographic data collection tool implemented at the beginning of the CS1 course allowed the researcher to identify persons in this class who had not taken part in the modified CS0 course. Persons attending CS1 during this term who had not attended the treatment CS0 course were not included as members of the control or treatment groups.

There were a number of potential outlier instances. These individuals were treated similarly to control and treatment group members and were allowed to participate. However, demographic information was collected to allow the researcher to identify these individuals so that data could be properly partitioned. Persons who did not attend a CS0 course at any point prior to the programming course were identified and placed in their own partition. Similarly, persons in the treatment CS1 course who were not participants in the control curriculum were partitioned into their own group. These outlier groups were expected to provide interesting supplemental information, but were not the focus of the research question. The remaining participants formed the proper control and treatment groups.

3.3 Research Design: Variables

3.3.1 Independent Variables

3.3.1.1 CS0 Curricular Alterations

The goal of this research was to isolate the impact of specific curricular modifications in the CS0 course. These modifications focused on improving the coverage of problem-solving methods and algorithm development techniques. The research design isolated this independent variable (changes to the CS0 curriculum) as much as possible within the constraints of working within a traditional educational domain.

In summary, the curriculum changes to the pre-programming course were:

1. Problem-solving Techniques
2. Diagramming techniques
3. Test plans for algorithms
4. Abstraction and modularity concepts
5. Collaboration methods

These changes may be viewed in detail in the Appendices of this document. Appendix I contains all readings, examples and exercises developed by the researcher for use in the CS0 course. Additional information for CS0 instructors, as well as information for participating CS1 instructors can be found in Appendix VII and similar information for students resides in Appendix VIII. This section provides a general description and justification for each modification. The existing curriculum was strongly related to the materials found in the Schneider and Gersting text: *An Invitation to Computer Science* (1999).

Existing CS0 Curriculum (Control)	Additions to CS0 Curriculum (Treatment)
Introduction to Computer Science Ch 1 text	
	Introduction to Problem-solving New reading, examples, exercises
Algorithms and Pseudocode Ch 2 text	
	Algorithms and Diagramming New reading, examples, exercises
	Algorithms with Pseudocode and Diagramming: new exercises
	Testing Algorithms New reading, examples, exercises
Complexity of Algorithms Ch 3 text	
Hardware Ch 4 text	
Organization Ch 5 text	
	Team Development New reading, exercises, examples
Ethics Ch 13 text	
	Software Engineering & Process Diagrams New reading, exercises, examples
Closing Materials	

Table 2: Curricular Modifications**3.3.1.2 Problem-solving**

The problem-solving unit bridged the gap between non-computing related problem-solving strategies and computing problem-solving strategies. As an introductory course for the major, it was very important that participants be reacculturated to a new way of thinking and working (Bruffee, 1990). This section attempted to link common situations and problem-solving with computer science problem-solving. In particular, the problem-solving process was adapted in the additional curriculum to mirror a standard generic computer science process:

1. *Understand and isolate the problem*
2. *Brainstorm for ideas to solve the problem*
3. *Design a solution that might work*
4. *Test your solution to see if it will work*
5. *Assess whether the solution is good enough to do it*

This set of steps can be easily mapped to requirement, design, implementation, and test cycles found throughout computing curricula. However, this process was devoid of terms that necessitated definition and could be applied to problems that had little need of excessive explanation, which allowed the student to focus on the process, rather than on rote memorization of multiple terms and rules.

Examples were provided in order to reinforce the processes outlined in the reading; exercises were also provided to encourage action in learning. Individual instructors were encouraged to implement each of these during the class in some fashion, although exercises were not anticipated nor required to be part of the graded materials for the class. In this section, the examples consisted of either simple or entertaining problems, which addressed the likelihood that there would be a wide range of ability in each group of students. The entertaining yet difficult problems were intended to appeal to those with greater ability and a desire to move forward quickly. Simpler problems that required less explanation reduced distraction from the intended focus of this component of the curriculum.

The duration of this section was brief and intended to provide a useful bridge to the development of algorithms section. It was hoped that this material would provide either a sufficient analogy or cement fundamental understanding in a way that would make formal representations of solutions more possible.

3.3.1.3 Diagramming

The existing curriculum relied primarily on formal definitions and the use of pseudocode in order to introduce students to the development of algorithms. The treatment in the text was very efficient and focused on showing examples and encouraging active attempts by the reader to write pseudocode. Additional material provided for diagramming followed a similar structure and was consistent in its presentation of control structures (sequential, iterative and selection). Rather than build material that was duplicative in nature, the new diagramming section focused on introducing the tool and encouraged active exploration with a set of twenty-five exercises.

This section was inserted in the curriculum in order to achieve two purposes. First, it was hoped that the addition of a pictorially based representation of algorithmic solutions would address multiple intelligences in the learner. In particular, this additional tool was expected to be useful for those who learn best from visualizations rather than text. Second, the extra effort of encouraging students to try both diagramming and pseudocode supported repetitions in learning while still introducing a new tool. Simple repetition is often met with impatience, but redundancy couched inside of new learning is important to show linkage and importance to the learner. Thus, students were exposed a second time to control structures and the refinement of steps into atomic units, which accomplishes some of the 'spiraling' technique discussed in chapter two.

3.3.1.4 Testing

The testing section provided further emphasis on the tools of algorithm development by providing additional examples and exercises for developing pseudocode and diagrams. In fact, examples were evenly split between diagrams and pseudocode and exercises encouraged mapping from one technique to the other. The curriculum presented here also

began to impart the concepts of testing into the mind of the learner. In particular, the value of testing was emphasized, as was the concept of successful tests being those that find problems (rather than a test that finds none). Examples illustrated how simple problems can lead otherwise intelligent problem-solvers into making common mistakes.

This section also contained a limited set of reading material, but sizable sections of examples and exercises. The content at this point became more technical in nature and included a greater number of terms and concepts. However, problem sets still utilized material that was easily understood and could be explained quickly to the student. Thus, this new piece of the curriculum successfully provided another iteration on algorithm development while providing new and useful information that could be applied at later points in learning.

Furthermore, this section began to encourage students to look critically at proposed algorithmic solutions by checking for accuracy. The next section concentrated on algorithm complexity and efficiency, which is also a process of critically assessing solutions. However, the prior curriculum did not provide material to emphasize correctness, so this section of material introduced the students to critical analysis of algorithms by using the simple idea of whether the solution will output results that mirror correct solutions. This was expected to result in an increased willingness to explore an algorithm more carefully after its creation, opening students up to the idea that how well the correct solution is accomplished is also important.

3.3.1.5 Team Development

The team development section was intended to begin to prepare students for the introduction of team projects in future computing courses, as well as the inevitable project team in the “real-life” development of computing

projects. This section provided a very brief summary of roles often found in the real world projects in order to emphasize the applicability of these concepts. It then quickly refocused on how one can successfully work in a group as a student in a class. Simple concepts covered in the reading are as follows:

1. *Strive for continuous improvement versus delayed perfection!*
2. *Avoid the Hero Syndrome*
3. *Pay now, instead of later.*
4. *Paper trails - being followed can be a good thing.*
5. *To collaborate, you must communicate*

Each of these sections encouraged equal and consistent participation of all members of a group. Problem design and problem-solving again made an appearance, as did the documentation of solutions and decisions (in the form of diagrams or algorithms). A discussion on conflict within the group was included as a reminder that there is a difference between positive and negative conflict. Rather than relying on a class lecture on this subject, an in-class exercise was provided to the instructor. The exercise encouraged problem identification and prioritization as well as solution negotiation. Communication skills were exercised as the activity was broken down into a 'jigsaw' formation, which encouraged each individual to be able to explain and discuss details of their subtopic. This course segment was intended to last no more than one class period and was built to lead into the ethics portion of the class.

3.3.1.6 Abstraction and Modularity

This section provided students with an introduction to software development language and processes prior to actual attempts to program. This section should be characterized as the logical equivalent to the detailed sections on hardware, data representation, and machine

organization that appeared in chapters four and five of the text. Although much background was provided for hardware and discrete structures, little attention was given to this topic in the existing curriculum. It made sense to include an introduction to software engineering or software development processes, since the following course in the curriculum focused on building software by programming.

This section paralleled the problem-solving and algorithm development sections by renaming the five-step problem-solving process in terms typically found in software engineering circles. Organization for larger problem solutions was covered in this section and process diagrams were included in order to provide a useful visualization tool for modularization. Important software development concepts and terms (such as cohesion, abstraction, scope, and modularity) were included in this portion of the materials. Top-down (divide and conquer) and bottom-up problem-solving methods were outlined; the exercises included problems that illustrated these approaches, as well as cohesion and abstraction.

3.3.2 Dependent Variables

The dependent variables were defined as a result of the alternate hypothesis and the two subordinate hypotheses. Each of the subordinate hypotheses dependent variables are discussed prior to the alternate dependent variable since these were used to support conclusions about the more general variable.

3.3.2.1 Satisfaction in the CS0 Course

One of the two subordinate dependent variables was the measure of satisfaction students had with the CS0 course. It was hypothesized that the independent variable, consisting of the changes outlined in section 3.3.1 for the curriculum, would positively impact satisfaction with respect to the applicability of the CS0 course. This satisfaction was measured at

the beginning and the end of the CS1 course in order to determine if additional exposure to the subject area provided participants with a different perspective for the CS0 course.

3.3.2.2 Self-Confidence in Skills

The second subordinate dependent variable focused on self-evaluation by participants of their ability to program, problem-solve, perform mathematical calculations, and work with a computer. In particular, this variable was concerned with measuring student confidence in programming skills. Measurements were taken at the beginning and end of the programming (CS1) course, which provided a baseline data point that could be used to determine growth in confidence by participants.

3.3.2.3 Programming Ability

Programming ability was the primary dependent variable for this research project. It was hypothesized that students would learn to program better if they were exposed to the new curriculum materials than if they were not, so programming ability was measured by collecting data points at the beginning of the CS1 course in the form of a pretest. Since participants could not be expected to program at that point in time, the test consisted of problems that could be solved using words, pseudocode, diagrams, or other techniques known to the student. A second set of data points was collected at the end of the CS1 course in the form of a posttest. This test included four programming problems of varying difficulty, allowing participants to provide evidence of their skill at the end of the course.

3.3.3 External and Environmental Variables

3.3.3.1 Instructional Differences in CS0

Participating instructors were given new materials and a suggested outline for coverage after they were consulted about the existing course format. It was agreed that these materials would be taught with the intended order of integration into the course as a whole. On completion of the term during which the treatment group attended CS0, instructors of this course were allowed to do with the curricular material as they saw fit. However, during the treatment term, teachers followed the structure, used the examples and exercises and provided students with the resources developed by the researcher.

Curricular changes do not necessarily imply changes in teaching method by the person facilitating the course. While it is clear that participating course facilitators used the new materials and followed the provided format, it is impossible to conclude that presentation by one instructor would match that of another instructor. Since there was certainly variation in the actual application of this curriculum, there may have been variation in the measured effectiveness of the curriculum. Fortunately, the same individual was responsible for teaching the CS0 course for both the control and treatment groups at each school, so differences in instruction were largely isolated to the curricular changes themselves. However, there was still a possibility that the instructor taught the course differently, especially if the instructor was excited by the new curriculum. In that case, students might have performed better because of this enthusiasm displayed by the instructor, more than any other reason (Cook, 1967).

3.3.3.2 Instructional Differences in CS1

The difference in CS1 instructors had more potential impact on this study than possible differences in the CS0 instructor. Two different instructors

taught BSU CS1 courses, with one teaching the control group and the other the treatment group. The CS1 curriculum was similar for both terms, but instructor implementation may have differed significantly. MSU courses were taught by four different participating instructors, with several other non-participating instructors teaching concurrent courses. In this case, there was a difference in instructor between control and treatment groups, as well as a difference for members within these groups. This study measures programming skill and how it is impacted by the CS0 curriculum; therefore, other factors that might influence the growth of this skill cannot be discounted. For example, an excellent CS1 instructor may facilitate better learning in one class as compared to a poor instructor in another. It was possible that instructor ability could sway the results of this research in favor of either the control or the treatment group, and thus obscure the true impact of the curricular changes (Black and Deci, 2000).

Although it was possible that measurements would be skewed by instructor differences, the triangulation in the research design was expected to limit the impact of this external variable. Student evaluation of the CS0 course was independent of the instruction provided in the CS1 course. In other words, it was unlikely that different instructors would greatly impact student opinion about the prior course. Furthermore, open-ended questions in the demographic data collection tool and exit survey provided participants with an opportunity to suggest factors that had strong impact on their learning. It was anticipated that exceptionally strong or poor teaching performances would be noted in the qualitative analysis. However, it was deemed likely that instructor quality would be within the range of typical norms and that it would not be sufficient to overcome data agreement for hypothesis and subordinate hypotheses.

3.3.3.3 Class and Term Dynamics

Each student group has a unique dynamic that is a product of the participants, the facilitator, the time and the environment (Brookfield, 1991). Therefore, it is impossible to claim that any two groups in a study of learning are equivalent, just as it is impossible to claim that any two learners are equivalent. Some groups may have been more open to the new curriculum than others, and some groups may have simply been more able to succeed in learning programming than others. This could have been true even if both groups contained individuals with relatively similar potential for success.

The nature of the sample provided this research with a representative group by taking a time-based cross section of the population. This method of sample selection created an effectively random sample on which standard statistical tests could be made. Therefore, differences in group dynamics were handled primarily by randomization of the sample.

Characteristics between student groups during fall and spring terms could also have differed depending on program structure. Frequently, programs will see the majority of entrants in the fall term, with a reduced set in the spring term. Sometimes, the second set of students consists of persons who are less certain of their ability to perform in the program. Furthermore, fall terms are usually preceded by an extended summer break, which may affect the retention of prior learned concepts. Neither BSU nor MSU exhibited drops in enrollment that would indicate that this sort of issue held for these schools. However, as other natural differences brought about by different terms could still have impacted the research data, a triangulated study was undertaken in an effort to address this external variable.

3.3.3.4 Learner Preferences and Background

Students bring with them a set of learning preferences and a set of experiences from which they can draw as they learn (Eble, 1990). The personal nature of learning makes it extremely difficult to generalize the success of any treatment intended to supplement learning (Diseth, 2003). It is even possible that such generalization is detrimental to learning in many cases. However, this study has been undertaken in order to identify tools and concepts that provide a valuable foundation for learning programming, which is a key component for any computing degree program. The personal nature of learning was addressed by collecting data at two points in time and pairing these results. This enabled the researcher to focus on areas of growth for each individual, rather than forcing all individuals to be compared with a norm.

3.3.3.5 Environmental Variables

Other environmental variables (such as the classroom, time of day, and class size) can have a decided impact on learning (Wilson and Shrock, 2001). These variables are also outside the control of the researcher. Class sizes between control and treatment groups were similar, but no data was collected on other factors. Triangulation and the cross-sectional nature of this study help to offset some of these factors, as does participation by several instructors with a varied set of times and classrooms.

3.3.3.6 Study Deployment Variations

A complete set of instructions, including a boilerplate introduction, was given to each CS1 instructor for deployment of this study. However, the researcher was not able to oversee this effort directly and contact was maintained with participating instructors from a distance. There was a distinct possibility that the implementation of the data collection tools would be undertaken differently in each class. Multiple data collection

points, method triangulation, data point redundancy, and full cross sectional participation were expected to work to counteract these issues and to reduce validity and reliability concerns.

3.4 Research Design: Data Collection Tools

3.4.1 User Identification Numbers to Support Paired Analysis

Each data collection tool included a clearly marked data entry point at the top for the participant to enter their student identification number. These numbers were used in lieu of names or characteristics that could inadvertently identify individual students to the researcher. Correct entry of the identification number for each of the four data collection tools was necessary for a full instance to be recorded in the research data. Correct entry of the identification number was critical for the success of this study, since failure to provide correct identification would result in data points that could not be linked under a single student. This would produce posttests with no matching pretest, demographic data with no matching qualitative data, and other mismatches. Failure to pair data points would lose information necessary to identify outlier status, so it would not be possible to conclude that data belonged to a member of a control or treatment group with any certainty. Therefore, it was necessary to remove any recorded data records that could not be conclusively matched with corresponding data collected by other collection tools.

3.4.2 Demographic/Entry Data Collection Tool

The demographic, or entry, data collection tool served multiple purposes in data collection. Demographic information for each participant was collected at the beginning of the CS1 course with this tool. Among the demographics were necessary questions to determine whether the participant was a member of the control, treatment, or an outlier group. Baseline information was taken for each individual's rating of their own abilities so these ratings could be compared with a second self-evaluation

at the end of the course. Similarly, participant opinions with respect to the CS0 course were collected at this point, and again at the end of the course in the exit survey. Finally, open-ended questions were used to gather supplemental information about participant opinions regarding the CS0 course and their anticipation for the upcoming CS1 class. Samples of this data collection tool can be found in Appendix II.

Demographic data points were selected for collection based on one of two reasons. The first reason was to illustrate that the sample population was representative of the general population of potential students that would participate in a computer science degree program at institutions similar to the subject universities. The second purpose was to collect information on characteristics that might influence individual success in a computing degree program. This information was collected once, at the beginning of the CS1 course, as a part of this data collection tool. Data was collected solely by self-disclosure and secondary tools were not used to verify this information.

Standard demographics such as gender, race, and age were collected to gain a picture of the diversity within the groups. This information was used to determine whether the sample was typical of most student groups at the subject institutions. However, this data was also used to partition other data points to see if a particular gender, age, or race exhibited greater change than other subgroups.

Demographic data related to the individual's academic status was also collected. Data points included year in school, typical high school grades, college grades, and the grade received in the pre-programming (CS0) course. This information provided a baseline status for each student, and also produced information that could be used to develop typical student profiles at this point in the program. This data was used to partition

instances based on academic experience. For example, these pieces of information made it possible to analyze whether the CS0 grade was a predictor for success in the CS1 course. Persons who did not complete a CS0 course were asked to circle "Not Applicable." Individuals who did report a CS0 grade were asked to identify the semester and year of completion. This information allowed the researcher to identify any outliers in the control or treatment groups.

Environmental or personal demographic data that could potentially impact learning was collected by soliciting responses on disabilities and competing activities. Participants were asked to indicate whether they were parenting, commuting, or employed outside of school. Each of these pieces of information might indicate competition within the individual's life for time or energy that could otherwise be spent on the course of study. The subject was also asked to identify any learning or physical disabilities that provided an additional challenge to their success in the course. This information was gathered in an effort to identify possible overriding factors that could have impacted learning. No strong patterns were exhibited for these data points, so little work was done with them in analysis.

Finally, data were collected on the participant's plans for future study in computing sciences. This information was gathered solely to determine if the sample population did represent a population that expected to continue with studies in the program. Anticipated future study indicated that the student expected to apply concepts learned in these courses in the future. If no such pattern existed in the population, it would have become apparent that the sample was not representative of the group the study hoped to research.

3.4.3 Self-Evaluation

The self-evaluation portion of the data collection tool consisted of five questions that collected ordinal data. Each question asked the participant to rate their level of comfort with a different task or concept. A value of 'one' indicated that the individual felt the task or concept was very difficult and a value of 'five' indicated that they felt it was very easy. Three of these questions measured comfort in working with tasks or concepts that are traditionally felt to be parts of the foundations of programming and computer science. The final measurement attempted to determine the individual's baseline comfort with programming.

Rate your level of comfort with computers

Each participant was asked to indicate (on a scale of 1 to 5, with 1 being low) how he or she felt about using computers. It was anticipated that most subjects would respond with an average to high level of comfort with computers upon entry to the CS1 course. Individuals beginning the course with a low baseline score were indicating that they believed they have an additional learning curve to climb in addition to course objectives.

Rate your level of comfort with math

The link between computer science and mathematics is both historical and conceptual in nature. Although it was expected that a majority of respondents would indicate average or higher comfort in this area, there has been an observed trend towards more computer science students who have less comfort in math skills. This data point provided the researcher with a method of determining whether or not a partition might be indicated by comfort in mathematics for programming skills. It was deemed possible that persons with weaker math skills might benefit more from the treatment curriculum. However, it was also considered possible that poor mathematical skills might have a stronger correlation with

programming success. Therefore, this data point was included in the data collection process.

Rate your level of comfort with problem-solving

One of the contentions of this study is that problem-solving skills lead to an increased ability to learn how to program. It was expected that most individuals would identify themselves as being average or better problem-solvers. However, it was considered possible that members of the treatment and control groups might identify this category differently. A significant difference in the numbers for this attribute was viewed as an indication that there was at least some impact on self-perceived problem-solving skills.

If you have programmed before, rate your level of comfort with programming

Respondents were instructed to leave this question unanswered if they had no programming experience to report. It was expected that a significant portion of respondents would leave this question blank on the demographics data collection tool. However, answers to this question provided information about prior knowledge owned by individual members of the learning group. Of particular interest would be any partition of subjects who identified no prior knowledge or very poor comfort with programming against those who declared strong comfort with programming at the beginning of the class.

3.4.4 CS0 Satisfaction

How useful do you think CS0 will be for your success in this (CS1) class?

This question was also rated from one to five, with one being low. A low score was classified as a 'waste of time' and a high score as 'extremely helpful.' This data point measures satisfaction by equating it with perceived applicability. Data collected at the beginning of the CS1 course

provided a baseline, as it was deemed unlikely that subjects would have sufficient information to accurately gauge the value of the prior course for one they had barely begun. However, this information was important to gauge initial perceptions of satisfaction so that any changes could be identified. Persons who did not complete a CS0 course were instructed to omit an answer to this question.

3.4.5 Qualitative Data

Three open-ended questions were included in the demographics data collection tool. These questions were intended to capture supplemental information that could be used to support patterns found in the quantitative data.

What were the most useful and least useful parts of the CS0 course?

This question was effectively paired with the rating of satisfaction provided in the prior data point. The wording of the question was intended to bring out free response about parts of the prior course that were perceived as helpful or as a 'waste of time.' It was anticipated that persons who rated the course as being wasteful would have stronger feelings for the least useful portions of the pre-programming course. It was also hoped that patterns in responses might indicate subject areas that were perceived as more or less useful. Pattern changes between the control and treatment groups were viewed as potential support for other data.

What are you most looking forward to in this (CS1) course?

What are you most worried about in this (CS1) course?

These two questions were intended to capture any patterns in the expectations and anticipations of students. It was expected that most commentary would include references to grades or learning to program. However, this data was collected with the knowledge that special circumstances, unknown to the researcher, could be revealed in this

fashion. If a strong pattern appeared out of the norm in these responses, it would be deemed likely that the sample was not representative of the target population.

3.4.6 Exit Survey Data Collection Tool

Samples of the exit survey can be viewed in Appendix III.

3.4.6.1 Demographics

The only demographics collected by the data collection tool pertained to the respondent's intentions to continue to study computer science on completion of the CS1 course. This data served the purpose of determining whether the sample remained representative of the target population by showing that members still intended to continue with studies in computing. However, it also served an additional purpose for the participating institutions; it gave them a picture as to how the CS1 course changed program of study intentions.

3.4.6.2 Self-Evaluation

The same four questions were asked in the exit survey so they could be paired with the results in the demographics tool. The only difference was a rewording of the fourth question: the qualifier "if you have programmed" was removed. The only rating expected to show significant change was the self-evaluation rating for programming. Although it was considered possible that other marks would exhibit change, it was not expected to be significant.

3.4.6.3 CS0 Satisfaction

The satisfaction question was worded similarly to that found in the demographics data collection tool. A second data collection point measured perceptions of applicability after the participant was more completely aware of the needs exhibited in the CS1 course. A severe

decline in satisfaction would have indicated that anticipated usefulness was proven to be unfulfilled. This might also have been an indication that the CS0 curriculum did not, in the opinion of the student, support the needs of the CS1 student. On the other hand, a less pronounced decline would have served as an indication that the subject had become more critical with additional knowledge, but that the CS0 module withstood a more critical test once the student had obtained more complete knowledge.

3.4.6.4 Qualitative Data

Qualitative information in the exit survey was collected from the perspective of reviewed events, rather than that of anticipation for new events (which was the entry tool's goal). The first two questions remained the same. It was expected that patterns for most useful and least useful parts of the CS0 course would adjust based on improved ability to see what tools were actually helpful in the CS1 course. The second two questions differed in both style and content.

Is there anything that you think should have been covered in the CS0 course that would have helped you in this programming course?

Do you have any suggestions that can help us to improve the CS0 or CS1 courses?

These questions were intended to encourage open-ended response by students as to how they might improve the CS0 course. It was anticipated that a significant subset of participants would be willing to share useful insights about this course when asked. However, it was also anticipated that subjects would be inclined to keep answers short due to time constraints in data collection. Significant patterns in responses to these questions were intended to serve as suggestions for additional curricular

reform, or to contradict or support the reforms implemented for the treatment group.

3.4.7 Pretest

There were two versions of the pretest provided to participating instructors. Only the first and last questions differed in these pretests. These versions were distributed alternately to participants and results of both versions were checked for consistency. Samples of the pretests may be viewed in Appendix IV.

Question 1: Algorithmic Problem-solve

The algorithmic problem-solving problem consisted of a problem that required a combination of selection and comparison operations in order to test participant ability to identify all options and choice paths. The actual problem differed between the two pretest versions, although each included multiple leaves on a solution tree. Solutions for this problem required some sort of clear organization for clear presentation of the proposed process. Therefore, it was expected that pseudocode or diagram representations would provide some of the more concise definitions of proposed solutions.

Persons completing this problem by identifying all possible options and providing an algorithmic solution that correctly produced all possible answers were given a full score out of five points. Absence of one or two obscure cases resulted in a rating of four points. Absence of multiple cases within a reasonable framework or structure for the solution resulted in an average score (three of five). Persons with the beginnings of a reasonable solution received a two and persons with hints of useful work received a one. Failure to work on the problem resulted in a zero for the question.

Question 2: Mathematical Problem-solve

The mathematical problem-solving question asked the participant to convert a number given in minutes to a representation given in years, days, hours, and minutes. The problem used a subject that was expected to be well known to all participants and to require no special knowledge beyond the ability to understand common time partitioning units. This problem exercised mathematical problem-solving abilities and also checked for the subject's awareness of exceptions (such as a leap year). This problem additionally exercised the ability to work with variables and understand how variables change or stay the same depending on the operations selected. It was anticipated that this problem would require some sort of structured algorithm representation in conjunction with mathematical calculations in order to clearly demonstrate a successful solution.

As before, a correct solution that clearly demonstrated steps to arrive at the answer, regardless of possible exceptions, received a perfect score. If the solution exhibited some confusion about numeric data carried through in a variable, the answer was given a score of 'four.' Failure to properly carry through the remainder to calculate new amounts resulted in an average score, and a reasonable structure for the solution with no working components received a 'two.' As with other questions, a single point was awarded for minimal effort and a 'zero' for no effort.

Question 3: Large Problem-solve

This problem asked the subject to outline a strategy for building a program (without actually programming) to play a game that they know (examples of chess, checkers, go, and bingo were given). The intent of this question was to ascertain relative strengths in determining scope and organizing complex problems. It was expected that most participants would perform an average to poor job of attempting to outline the problem domain and

create a reasonable strategy for the problem. However, the question was included in order to help identify those with exceptional skills. Better solutions were expected to use a combination of algorithms, diagrams, and words to illustrate structure, scope, and process.

An average score for this problem was awarded for those who successfully managed to provide a clear strategy from one perspective of the problem. For example, a potentially successful functional breakdown would receive an accurate score. Additional marks were awarded if other perspectives in identifying and strategizing a solution supplemented the single perspective. For example, a functional breakdown that included some attempt to identify data structure would receive additional marks. A score of 'two' indicated that a reasonable attempt to organize the problem was made and a 'one' indicated that a poor attempt was observed. A 'zero' was reserved for those who made no effort to answer the problem.

Question 4: Brainteaser Problem-solve

A brainteaser question was included to test the participant's ability to identify a problem and its key components. The solution to this problem was not intended to be presented as an algorithm, although subjects were encouraged to show all ideas and thoughts that they tried to use in solving the problem. It was anticipated that answers would be accompanied by text and graphical representations to either attempt to solve or explain the solution to the problem.

Persons with the correct answer, regardless of supporting documentation, received a perfect score for this problem. The actual distance from the correct answer was used to determine the score for the problem. Answers with incorrect solutions but correct processes received additional credit. Processes with no answer also received credit for the solution. Only

problems with no effort to show a process or a reasonable answer were given a 'zero' score.

3.4.8 Posttest

The posttest included four questions that called for answers written in a programming language. Participants were encouraged to show any tools or illustrations that helped them write their code samples. None of these samples were intended to be fully working programs and simple formatting mistakes were ignored in assessing answer accuracy. Examples of the posttest may be viewed in Appendix V.

Question 1: Simple Swap

The first question was selected based on the ease with which it could be explained to the participant and the expected likelihood that most subjects could find some success in writing code to solve the problem. Answers were to be couched in a function (C++) or a method (Java). However, correct solutions without these trappings still received high marks (4 of 5 points). Answers that failed to illustrate the ability to track the values of variables, but that otherwise exhibited a procedural structure that approximated the correct solution, were given three of five points. Varying degrees of effort that provided incorrect solutions were given scores of 'one' or 'two.' Persons who provided no evidence of an effort to solve the problem received a zero for this data point.

Question 2: Palindrome

The first question was selected based on the ease with which it could be explained to the participant and the expected likelihood that most subjects could find some success in writing code to solve the problem. Answers were to be couched in a function (C++) or a method (Java). However, correct solutions without these trappings still received high marks (4 of 5 points). Answers that failed to illustrate the ability to track the values of

variables, but that otherwise exhibited a procedural structure that approximated the correct solution, were given three of five points. Varying degrees of effort that provided incorrect solutions were given scores of 'one' or 'two.' Persons who provided no evidence of an effort to solve the problem received a zero for this data point.

Solutions that failed to handle the 'odd length' case, but were otherwise correct, received four of five points. Similarly, solutions that were not written as a function or method had their score reduced by a point. Thus, answers that were not in this format and that were missing the special case were rated as average answers at three points. Some effort in solving the problem received one point, while submissions that showed evidence of being at least partly correct received two points.

Question 3: Big X

This question required less effort to understand the intent and scope of the problem, but included more subtle nuances in both the algorithm and the program code. As with other problems, answers that had a structure that could house an entirely correct answer (provided they were ordered correctly or all cases were identified) received average marks. Ratings followed a similar structure to previous questions.

Question 4: Large Problem

The large problem was a companion to the large problem found in the pretest. In this case, the participant was asked to identify method/functions definitions and variables necessary to help a school schedule classes. Essentially, this problem asked students to identify the parts of a program necessary to solve the large problem without requiring that they write specific code segments. The problem did not require a great deal of explanation so that concentration could be maintained on attempting to solve the problem. Scoring this problem was based on

coverage of the scope of the problem, with an adjustment downward by one point if the code format for the proper language was not evident.

3.5 Responsibilities of Participants

Student Participants / Subjects

- ◆ Subjects were asked to listen to a description of the study and the informed consent boilerplate.
- ◆ Subjects read and signed the informed consent form.
- ◆ Participants were asked to complete all forms in their entirety during the time allotted and to provide honest and thoughtful answers.
- ◆ Students were asked to use their student identification number for pairing purposes on the data collection tools.
- ◆ Subjects were encouraged to contact the researcher or their instructor if they had questions about the study.
- ◆ Participants completed the demographics data collection tool at the beginning of the CS1 course.
- ◆ Students completed the pretest and were instructed to show as much evidence of their approach to solving the problems as they could.
- ◆ Participants completed the exit survey at the end of the CS1 course.
- ◆ Subjects completed the posttest and were instructed to show as much evidence of their approach to solving the problems as they could.

Participating Faculty and Departments

- ◆ CS0 instructors negotiated with researcher on content of material provided for pre-programming course.
- ◆ Participating CS0 instructors implemented course with curricular modification during Fall 2000 term.
- ◆ CS0 instructors reported inconsistencies or problems with the new CS0 materials in a timely fashion for correction and use during the Fall term.

- ◆ CS0 instructors were granted continued use of any and all new materials created by the researcher.
- ◆ Participating CS1 instructors agreed to remain in contact with the researcher during the Fall 2000 term (control group) and the Spring 2001 term (treatment group).
- ◆ CS1 instructors agreed to read the informed consent boilerplate to participants, collect signed informed consent forms and serve as first contact for participants in the study.
- ◆ CS1 instructors administered a demographic data collection tool and a pretest on the first day of the programming courses (CS1) in the Fall 2000 and Spring 2001 terms.
- ◆ CS1 instructors administered an exit survey and a posttest at the end of the CS1 course during the Fall 2000 and Spring 2001 terms.
- ◆ CS1 instructors delivered all documents (informed consent, pretest, posttest, demographic survey and exit survey) to the researcher.
- ◆ CS1 instructors were asked to report any requests to be removed from the study during the term.
- ◆ Departments were asked to support the researcher in completing the standard research review board process for each school.
- ◆ Departments were asked to confirm that the first programming course (CS1) would not undergo a major change in approach during the Fall 2000 and Spring 2001 terms.
- ◆ Departments were asked to confirm that the CS0 course was intended as the entry-level course for the major.
- ◆ Departments were asked to confirm that the CS0 curriculum had not significantly changed in ways different than those proposed by the researcher.

Researcher

- ◆ The researcher negotiated with faculty regarding the content modification to pre-programming (CS0) course.

- ◆ The researcher developed supplemental material for use in the pre-programming (CS0) course on those subjects agreed to by the participating instructors and the researcher. Material included notes, examples, and exercises that supplemented those provided by the Schneider/Gersting text.
- ◆ The researcher developed a demographics data collection tool, exit survey, pretest and posttest for use in this study.
- ◆ The researcher encouraged discussion and negotiation of data collection tool design prior to the Fall 2000 start date for the study.
- ◆ The researcher provided departments with proper research consent forms for student participation.
- ◆ The researcher maintained contact with participating instructors and provided resources for participating students.
- ◆ The researcher maintained the anonymity of participants in the study.

3.6 Review Processes

The anonymity of participants in this study was maintained by use of a student identifier, in lieu of a name, to link data points between data collection tools. Individual data was further protected by the substitution of a generated identification number once all data points were combined into single records. No identifying information appears in this document or its appendices, or will it appear in any other report related to this research.

Instructors were not given individual responses to the demographic tool or the exit survey, nor were the pretest or posttest used for grade assessment purposes. Participation or non-participation in this study had no bearing on student participation in either the CS0 or the CS1 course. The CS1 curriculum was not modified in any way to take advantage of changes to the CS0 curriculum, so students in the control or outlier groups were not disadvantaged by assumed knowledge built into the second course.

Each participant was given a human subject consent form at the beginning of the programming (CS1) course and an opening statement was read to participants prior to any data collection. All members of participating CS1 sections were given the option of accepting or refusing participation at this time. Participants agreed to participate in the study when they signed the consent form and submitted it to the participating instructor, who then forwarded these materials to the researcher. Subjects were reminded that they could remove themselves from the study at any point in the process, although there were no recorded instances of persons exercising that option. A copy of the informed consent document may be viewed in Appendix VI.

Since this research involved human subjects, Institutional Review Board (IRB) processes were followed for each of the subject institutions at BSU and MSU. Additionally, the researcher submitted materials to the Union Institute and University (UIU) IRB for approval. Each of these boards gave unconditional approval for this research project. Submitted materials and approvals can be seen in Appendices IX (UIU), Appendix X (MSU) and Appendix XI (BSU).

4 Findings

4.1 General Observations

4.1.1 Identification of Outliers

Outlier instances were defined as respondents for which data was collected as a part of the CS1 course, but who for various reasons were removed from the sample set. In order to be a part of the sample, individuals were required to have completed the CS1 course and both the pre and post data collection tools for the study. Those failing to complete the CS1 course were eliminated since no end of class data had been collected to measure growth. Participants were also required to have attended a qualifying CS0 course. Those participants who had either not attended a CS0 course, or those who had attended a CS0 course at another institution or during a term prior to the time frame of the research, were identified as outliers. Finally, data records that could not be matched with corresponding entries for other data collection tools were removed from the sample, as failure to match records made it impossible to perform the necessary paired analysis.

Although some incomplete data records were omitted from analysis entirely, other outlier records provided useful supplementary information. For example, data collected for persons who did not complete the CS1 course were collected in an effort to see if there was a pattern corresponding to retention in the programming course. Data for persons who completed the CS1 course but did not participate in the appropriate CS0 session were also maintained. In the case of the Spring 2001 class, most members completed the treatment CS0 course the previous fall term. However, some individuals did not attend the anticipated CS0 course, making them, in effect, a second control group. Outlier records comprise a secondary focus of discussion in this report.

Outlier types for the Fall 2000 term included:

- Persons who completed CS1 and had no CS0
- Persons who did not complete CS1 and had qualifying CS0

Outlier types for Spring 2001 term included:

- Persons who completed CS1 and attended non-treatment CS0
- Persons who completed CS1 and had no CS0
- Persons who did not complete CS1 and had no CS0
- Persons who did not complete CS1, attended non-treatment CS0
- Persons who did not complete CS1 and attended treatment CS0

4.1.2 BSU Sample

The Bemidji State University (BSU) data records were matched by student identification number, which allowed the researcher to complete an accurate picture of results for this school. It was determined that a qualifying CS0 course for the control group could include participation during the previous academic year; however, earlier attendance disqualified the participant from the control group, since it could not be successfully verified that the curriculum was the same in prior years. Furthermore, an extensive gap between courses naturally precluded inclusion in the control group. Few students attended CS0 in a term other than the Fall 2000 term, and as these persons failed to complete the course, they were eliminated from the control group.

	Enrolled	Completed	Outliers	Sample Size (n)
Fall 2000	33	25	1	24
Spring 2001	29	21	11	15

Table 3: BSU Sample

As can be seen in Table 3, the enrollment for both courses was roughly similar, as was the completion rate (75.8% for the fall and 72.5% for spring). The greatest difference between the two samples was the composition of the outlier groups. Obviously, the definition for the control

group was broader, selecting candidates who had attended the pre-programming course during one of two terms. However, additional data records that might have been added by this expanded definition were eliminated by non-completion. Two individuals completed the end of term data collection tools, but were not present to complete the beginning of term data collection tools, which explained part of the difference between the samples. This data could not be used, as there were no informed consent materials for these persons on file; therefore, these materials were eliminated without data analysis.

	No CS0	Wrong CS0	no Pretest
Fall 2000	1	0	0
Spring 2001	1	3	2

Table 4: BSU Outliers Completing Course

Outlier records for persons completing the CS1 course were not numerous, with one in the fall and six in the spring, of which five were identified as persons who had not taken a qualifying CS0 class (see Table 4). Interestingly, the same number of persons (five) failed to complete from the outlier groups (see Table 5). This indicated that persons in the outlier groups (with the exception of the two without a pretest) were equally as likely to succeed or fail in the programming group.

	No CS0	Wrong CS0
Fall 2000	0	0
Spring 2001	4	1

Table 5: BSU Outliers Failing to Complete

During both terms, eight individuals failed to complete the CS1 course. In the control group, none of these individuals fell into an outlier class; all eight had attended a qualifying CS0 course during the previous academic year. In the treatment group, five of those who failed to complete the course either had no CS0 course (four) or had attended a non-treatment CS0 course (one). This left only three persons in the spring term who

failed to complete the CS1 course after completing the treatment CS0 course.

4.1.3 MSU Sample

	Beginning	Ending	Matched	No CS0	Wrong CS0	Eligible	Match & Sample
Fall 2000	43	36	5	5	4	34	4
Spring 2001	47	30	22	4	4	39	20

Table 6: MSU Sample

Unfortunately, multiple student identification numbers were given to Minnesota State University at Mankato (MSU) students. Not only did enrollees in these courses have a student identification number, they were also assigned a technology identification number. Additionally, some persons considered their social security number to be an appropriate identification number. Participating instructors described the need for students to consistently identify themselves with varying success. As can be seen in Table 6, only five of the thirty-six ending data records could be matched successfully to the beginning (pretest, demographics) records. There was somewhat better success during the spring term, but again eight of the ending records failed to match a beginning record. Failure to match records made it impossible for the researcher to determine whether a participant belonged in the control or treatment groups, so no meaningful paired analysis could be performed on data collected for the study from MSU. Although the treatment set reached a reasonable sample size, a treatment set without a corresponding control set was useless for paired analysis the MSU data were discarded. The remainder of the discussion will focus on data collected from the BSU population.

4.2 Demographics

4.2.1 Personal Demographics

Personal demographics collected consisted of gender, race, and age attributes. This information was collected to determine if both control and

treatment groups were representative of the normal population found in the CS1 course. It was also used to partition data in data mining sessions in order to look for patterns based on these attributes.

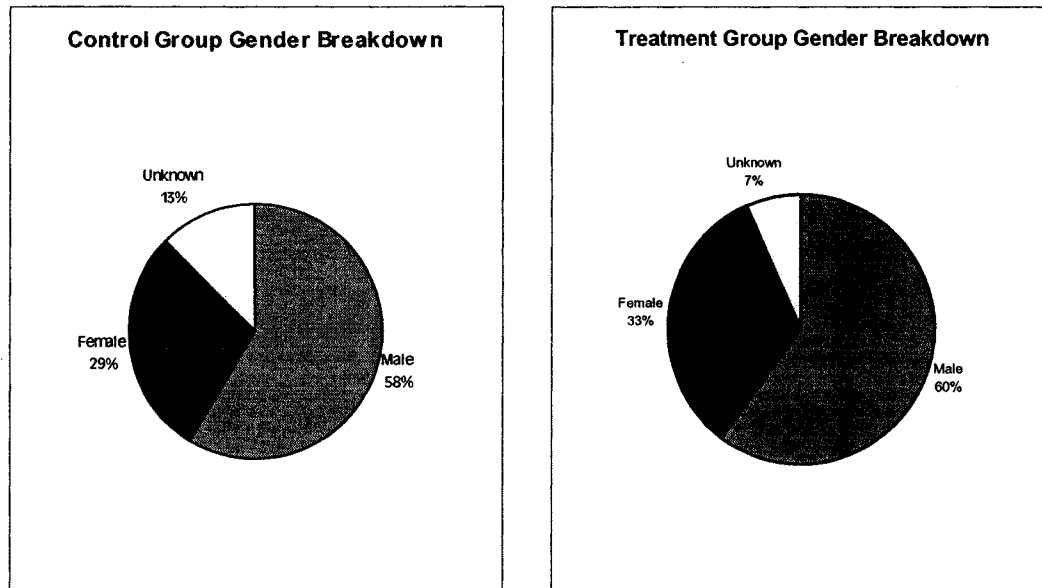


Table 7: Gender Breakdown

Gender distribution for both the control and treatment groups was roughly equivalent given the sample size. Table 7 gives a pictorial representation of this breakdown. (It should be noted that these charts only refer to instances that qualify fully for the treatment or control groups.)

As expected, the majority of the population for each group was male, which is consistent with nationwide enrollment trends for computer science programs. In fact, nearly twice as many participants identified themselves as male versus female in both the control and the treatment groups. Seven of twenty-four control instances were female, and fourteen were male. Three persons chose to omit their gender in the demographic data for the control group. Only one person chose to omit their gender in the treatment group; five were female and nine were male. Therefore, the

population appears to be consistent with general trends and across both the treatment and control groups.

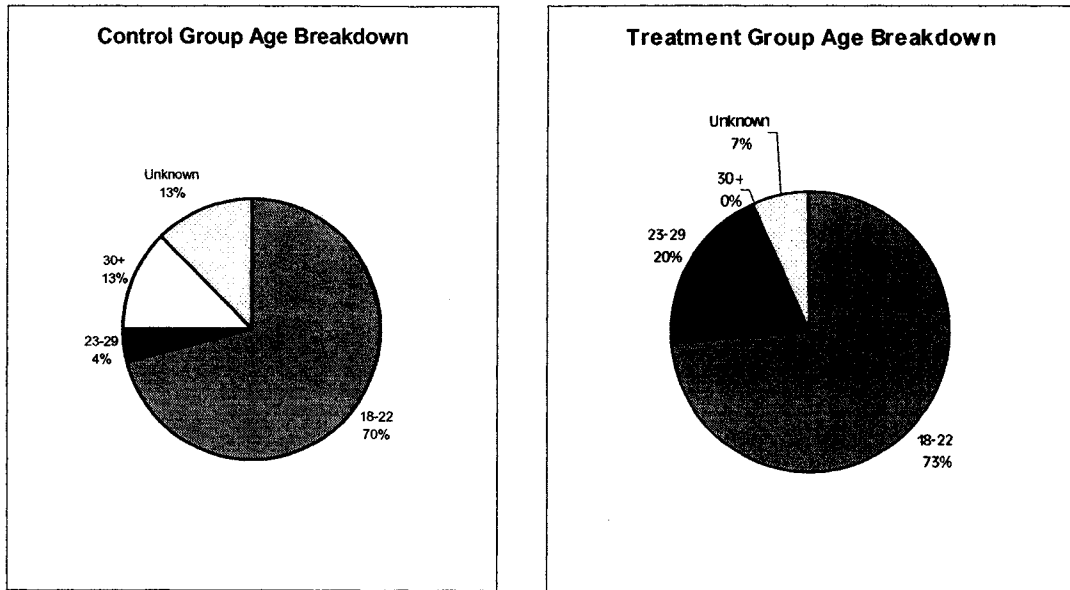


Table 8: Age Group Distribution

The breakdown between traditional aged and non-traditional aged students in both groups was nearly identical. A traditional-aged student was defined as being between the ages of 18 and 22, inclusive. Table 8 clearly illustrates that both groups consisted of a strong majority of persons in this age range (70% for control and 73% for treatment). The respondents who failed to provide gender identification also did not identify their ages on the demographics data collection tool. The 23-29 year old age group included individuals who had returned to school full or part time after several years work experience. Those over thirty tended to be changing careers and were often returning to school on a part time basis, so these groups were separated for possible partitioning. Three persons reported an age of at least 23 years in the treatment group; four persons did so in the control group. Three of the persons in the control group reported ages over thirty. Although this sample was small, it did

show consistency between traditional and non-traditional aged students in the two groups.

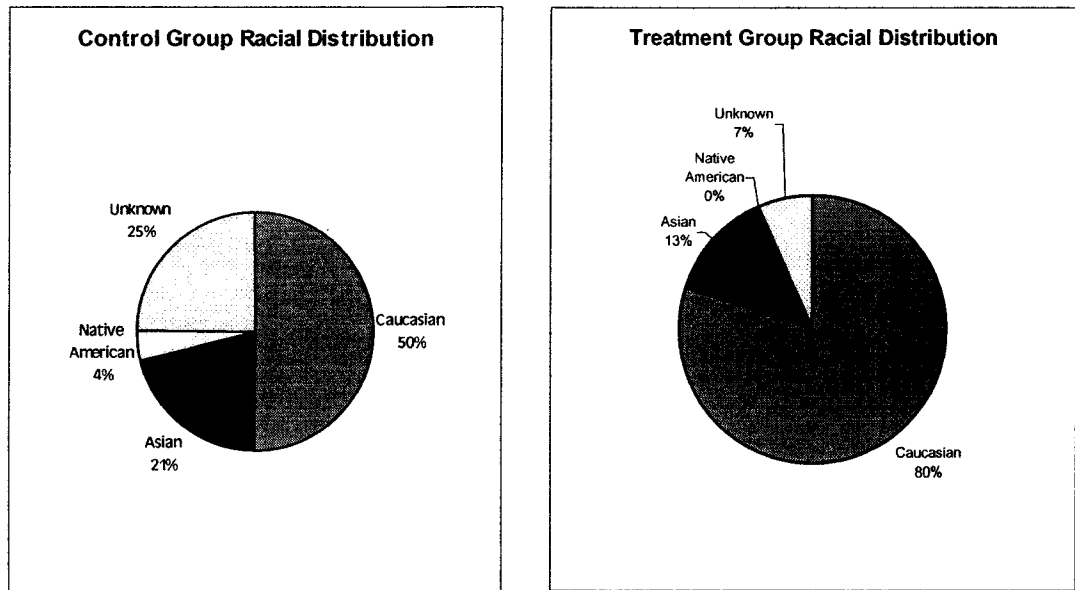


Table 9: Racial Distribution

A sizable portion (25%) of the control group subjects chose not to reveal their race in the demographic collection tool, so a comparison between the control and treatment groups revealed minimal information on data consistency. Table 9 clearly shows a Caucasian majority, which was expected for the region and school from which the data was collected. The largest minority population identified themselves as Asian in origin, with five participants in the control group and two in the treatment group. A single person identified him or herself as a Native American. The large number of unidentified attributes made it impossible to generalize as to whether the sample was representative of the population.

4.2.2 Academic Demographics

Academic demographics were collected to provide a picture of current progress in the degree program and academic achievement. Subjects were asked to report their current year in school, which reflected their

status at the point they were taking the programming (CS1) course. This information was used to partition less experienced students from more experienced students in data mining sessions. Participants were also asked to identify their normal expected grades for high school and college, as well as their awarded mark for the CS0 course. This data reflected the concern that prior success in academic environments was an external variable that might skew results.

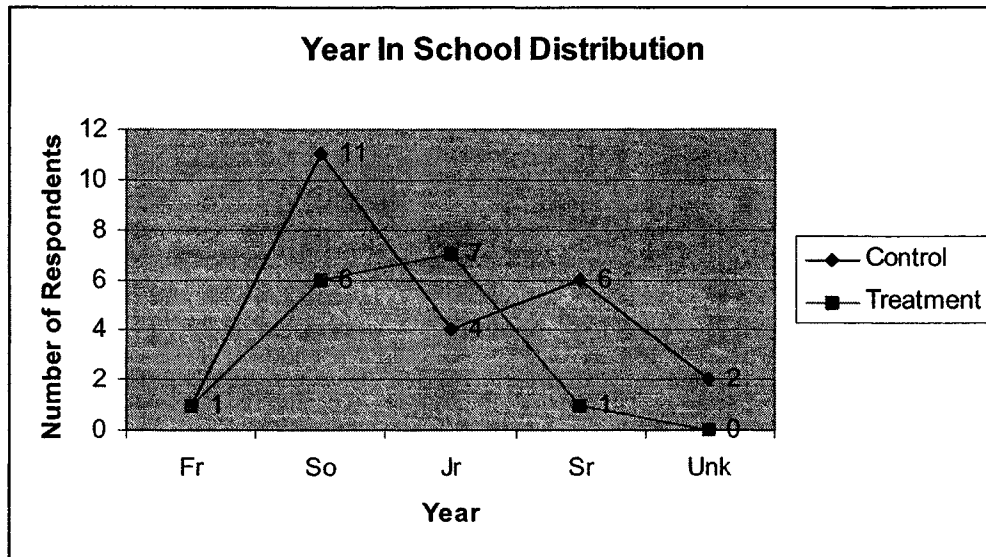


Table 10: Year in School

The distribution for participants and their current year of matriculation contained a surprisingly large number of upper class students. As shown in Table 10, the ratio of upper class to under class participants is nearly one to one. Neither section had a sizable enrollment of freshman, even when outlier instances were included. This distribution implied that younger students who expect to complete computing degree programs commonly took the CS0 course in the spring, which would typically be their second year of matriculation. The large number of sophomore students in the control group supported this conclusion, since this group would have participated in a fall offering of the CS1 course.

The large number of upper class participants raised the concern that many of the participants were not interested in continuing study in the field. This concern is addressed later in section 4.2.4. However, nearly all students identified themselves as majors or minors in computer science; therefore, many students in this program apparently expected to attend college for more than the normal four-year period. Also, it was considered likely that many participants had taken advantage of the college credit opportunities that are offered to advanced high school students at many colleges, which would account for a larger number of younger students who had greater college experience.

	A	B	C	Unknown/NA
CS0	7	6	8	3
High School	16	6	1	1
College	8	12	3	1

Table 11: Control Group Grade History

The grade history of the control group can be found in Table 11 and can be compared with the results for the treatment group in Table 12.

	A	B	C	Unknown/NA
CS0	7	6	1	1
High School	12	3	0	0
College	11	4	0	0

Table 12: Treatment Group Grade History

The data provided for grade history suggested that the group of individuals who completed the CS1 course in the treatment group had a greater history of academic success. High school marks were similar between the two groups, but the distribution changed in the college marks. However, the most marked difference occurred in the number of C grades received by participants in the CS0 course. Eight members of the control group reported a C, whereas only one in the treatment group reported a similar grade. This data indicated that there might be a population difference across the sample groups, which is discussed further in the analysis portion of this document.

4.2.3 External Learning Factors

External learning factors included characteristics reported by the student in the demographics data collection tool that may have directly, or indirectly, impacted success in school. This data is reported in Table 13 shown below:

	Part Time Job	Full Time Job	Parent	Commuter	Disability
Control	17	3	5	6	0
Treatment	12	0	2	1	3

Table 13: Reported External Learning Factors

A sizable number of participants in both groups worked part or full time off campus in addition to their efforts as students. This trend was consistent with the overall trend for college students at this school (BSU, 2000). The control group included a significant minority (five) who were parenting, and most of them also commuted to classes. Instances with these combinations were partitioned for further analysis in data mining. The treatment group included three individuals who identified physical or learning disabilities; these records were also partitioned to determine if these factors were significant in determining success or failure in the class.

4.2.4 Study Plan Demographics

	Major Minor Undecided			No Intent to Continue
Fall Pre	30	1	0	1
Fall Post	19	3	1	1
Spring Pre	20	3	0	4
Spring Post	17	3	0	1

Table 14: Study Plan Intentions

The high number of participants in their third and fourth college years was contrary to expectations for entry-level courses. However, each participant was asked to rate their intention to pursue studies in the available computing majors at BSU. Ratings of four and five indicated

strong intentions to continue their studies in the field; ratings of one or two indicated that the individual was unlikely to continue with studies in computing, and a three indicated indecision. As shown in Table 14, nearly all participants identified themselves as a major or minor in computer science, with a strong intention to continue those studies. This trend did not change significantly between the beginning and end of the course, although some persons had decided to alter their plans and acquire a minor instead of a major in the fall term. Of the four persons who showed no interest in continuing studies in computing for the spring term, three were identified as outliers who had not taken a CS0 course. In fact, those three persons also failed to complete the CS1 course. Therefore, it was determined that the population is reasonably representative of the typical BSU computer science student.

4.2.5 Outlier Instances

Fall 1	completed and no CS0
Fall 2	did not complete and CS0
Spring 1	completed and non treat CS0
Spring 2	completed and no CS0
Spring 3	did not complete and no CS0
Spring 4	did not complete and non treat CS0
Spring 5	did not complete and treat CS0

Table 15: Outlier Group Definitions

Outlier instances fell into one of seven subgroups, listed above in Table 15. Three of these groups consisted of persons who successfully completed the CS1 course, but were not participants in the expected CS0 course. The Fall 1, Spring 1, and Spring 2 groups consisted of instances where the participant successfully completed the CS1 course, while the other groupings included persons who failed to complete CS1. Data in this section were broken down by completion or non-completion; however, a breakdown by CS0 coverage may also be worthwhile in future analysis and study. Data were partitioned by those who did not take a CS0 course (Fall 1, Spring 2, and Spring 3), those who took an anticipated CS0

section (Fall 2 and Spring 5), and those who took an unexpected CS0 section (Spring 1 and Spring 4).

Outlier Group	Male	Female	18-22	23-29	over 30	Caucasian	Asian	Native American				
								Fr	So	Jr	Sr	
Fall 1	1	0	0	1	0	1	0	0	1	0	0	0
Spring 1	3	0	3	0	0	2	1	0	0	0	1	2
Spring 2	1	0	0	1	0	1	0	0	0	0	0	1

Table 16: Outliers That Completed Demographics

Individuals who completed the CS1 course yet failed to meet the criteria of being in a participating CS0 course were not common. In fact, of the six persons who attended CS1 with no prior CS0 course, only two completed it (Fall 1 and Spring 2 in Table 16). Of the four who failed to complete the course (Spring 3 in Table 17), three identified themselves as not being interested in continuing studies in computer science, which made the data for Fall 1, Spring 2, and Spring 3 groups uninteresting for the purposes of this research. Persons who attended a non-treatment CS0, but attended the CS1 course with the treatment group (Spring 1 and Spring 4) included three who completed the course and one who did not. All members of these groups had junior or senior status in the four-year program. These records were not tested for similarity with instances in the control group, even though they attended a similar CS0 course.

The most interesting groups in the outlier class were the Fall 2 and Spring 5 groups. Members of these groups fit the criteria for inclusion in the control or sample based on their attendance of the CS0 course. However, the individuals in these groups failed to complete the programming class, so no data existed for the posttest or exit survey. Even so, it was interesting to note that eight persons failed to complete in the fall group, while only three failed to complete in the spring. Although this information was not sufficient to reach substantial conclusions, it did illustrate varying levels of success by participants in each group.

Outlier Group	Male	Female	18-22	23-29	over 30	Caucasian	Asian	Native American				
								Fr	So	Jr	Sr	
Fall 2	6	2	4	3	1	5	1	1	0	1	4	3
Spring 3	3	1	2	0	1	4	0	0	2	1	0	1
Spring 4	1	0	1	0	0	0	1	0	0	0	1	0
Spring 5	3	0	2	1	0	3	0	0	0	1	2	0
Totals	13	3	9	4	2	12	2	1	2	3	7	4

Table 17: Outliers Failing to Complete

In general, the only demographic difference between the outlier groups and the control and treatment groups was the tendency for outliers to be students in their third or fourth year in the bachelor's program. Only six of twenty instances were classified as freshmen or sophomores, which differed from the nearly even break between upper and lower class participants in the control and treatment groups. Otherwise, the outlier group was primarily male, traditional college-aged, and Caucasian. The percentage of female outliers was lower than the instances of female participants who qualified as control or treatment group members (13% vs. 29% and 33%) and the number of non-traditional aged students was somewhat higher (40% vs. 17% and 20%). However, this may reflect individuals who failed to report these characteristics. In general, the outlier data were sufficiently similar to the non-outlier data to suggest that there was no bias against any particular demographic group in these courses.

	Part Time		Full Time		
	Job	Job	Parent	Commuter	Disability
Spring 1	1				
Fall 2	7		1		1
Spring 4					
Spring 5	2				1

Table 18: Outlier Reported Factors

Additional demographic factors reported in the outlier groups followed a similar pattern to those exhibited by members of the control and treatment groups. Table 18 shows reported factors for all outlier groups other than Spring 2, Fall 1, and Spring 3. The vast majority of participants reported

(whether they were in the outlier groups or not) that they were employed at least part time. A small number reported that they were parents, and a smaller number reported physical disabilities (most commonly eyesight) or learning disabilities (two instances of dyslexia). None of this information indicated that a distinct pattern for completion or non-completion existed for these data points.

	High School			College			CS0				
	A	B	C	A	B	C	A	B	C	D	NA
Fall 1			1	1							1
Spring 1	2		1			3			3		
Spring 2	1					1					1

Table 19: Grade History, Outliers Completing

The grade histories for outlier instances exhibited a few interesting differences from the control and treatment groups. High school grade history remained consistently high, as did most reported grade histories for college. An extremely small number of all participants reported an average C grade in the college experience, with three in the control group and one in the Spring 3 outlier group; therefore, it can be concluded that the vast majority of participants were accustomed to success in academics. Table 19 illustrates the grade history for outlier members who completed the CS1 course. The Spring 1 group consisted of persons who were taking CS1, but who had taken CS0 during a semester prior to the treatment term. All three persons reported a C grade, possibly explaining a delay in continuing studies as these individuals reassessed their desire to continue.

	High School			College			CS0				
	A	B	C	A	B	C	A	B	C	D	NA
Fall 2	7	1		3	5		1	2	2	3	
Spring 3	3	1		2	1	1					4
Spring 4		1			1						
Spring 5	1	1	1	1	2				2		
totals	11	4	1	6	9	1	1	2	4	3	4

Table 20: Grade History, Outliers not Completing

On the other hand, those who failed to complete the CS1 course reported a generally less successful experience, based on CS0 grades. In particular, those who failed to complete the CS1 course and qualified as potential members of the control group (Fall 2) reported the only D grades (3; see Table 20) for CS0. The Spring 5 group included students who completed the treatment CS0 course. This group included three instances, of which two reported C grades in the CS0 course. This accounted for one third of all of the C grades assigned to this group, which was significant given the fact that Spring 5 only included three of the possible twenty-nine respondents during that term.

These data, and similar data differences in grade history between the control and treatment groups, was considered in analysis. Students who achieve successful marks in a prerequisite may be more likely to complete the next course in the sequence (Stein, 2002). However, it cannot be said that completion of the course was accompanied by the reception of successful marks. Data were not collected for final grades in the CS1 course.

4.3 Pretest and Posttest Data

4.3.1 Coding/Marking Process

The researcher marked pretest and posttest questions, in order to offset any preconceived notions an instructor might have based on the historical success or failure of a particular student. In order to avoid researcher

bias, tests from both the fall and spring terms were mixed together and the term origin was blocked from view. All pretests were marked on the same day, as were all posttests. Once marked, the documents were regrouped by term and data were recorded.

There were four pretest questions, and three data points were recorded for each question. Pretest questions were evaluated for correctness and given a whole number score from one to five, with five representing a completely correct answer and one representing any attempt to solve the problem that resulted in very poor results. If no attempt was made to solve the problem no score was recorded. Thus, pretest results may show lower count numbers than actual participants for some question scores. In determining test totals, these blank instances were treated as zeros, but calculated averages for individual tests left these instances out of the sample. Each pretest question was also rated on a score from one to five for clarity. A very clearly presented solution would receive a high score (four or five), even if the answer were entirely incorrect (one or two). Similarly, a correct answer received low clarity scores if it was difficult to determine if the answer was correct and how the answer was determined. Finally, each pretest question was categorized based on the method chosen by the participant to present his or her answer.

There were also four posttest questions for which two data points were recorded for each question. All questions required answers written in the C++ programming language (the language used in the BSU CS1 course), with additional information being optional but encouraged. Each question was rated for accuracy with a value from one to five, with five being the highest. No fractional values were assigned and questions with no effort were assigned a zero score. The zero score was used both for the posttest total score calculation and for single question averages. If the participant showed evidence of supplementary processes to help with

coding, this was recorded as a second data point. Coding for this data point was done in a similar fashion to the third data point in the pretest. The type of representation of supplementary processes was identified and given a coded value equivalent to similar work exhibited in the pretest.

4.3.2 Control Group

4.3.2.1 Pretest

Participants in the control group left very few questions entirely blank, so some positive value was assigned for nearly every question. Of the four questions, the third was the most difficult, as it asked for a strategy for solving a large problem. As such, it was most frequently left blank (five instances; see Table 21). Most correctness ratings showed a normal distribution for the marks received by participants of the control group. The only possible exception to this was a slightly flat distribution for question four. Similarly, most clarity ratings showed a strong normal distribution.

	<u>Q1</u>	<u>Q1</u>	<u>Q2</u>	<u>Q2</u>	<u>Q3</u>	<u>Q3</u>	<u>Q4</u>	<u>Q4</u>
	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>
Answers	24	24	23	23	19	19	21	21
Mean	3.46	3.38	3.35	3.22	1.63	2.37	2.19	1.62
Std Dev	0.93	0.88	1.11	0.74	0.68	0.90	1.21	1.24
Variance	0.87	0.77	1.24	0.54	0.47	0.80	1.46	1.55
Skew	-0.40	-0.43	-0.34	-1.13	0.63	-0.34	0.35	0.99
Counts								
5's	3	1	4	0	0	0	0	1
4's	8	12	5	8	0	1	4	0
3's	11	6	11	13	2	9	5	4
2's	1	5	1	1	8	5	3	4
1's	1	0	2	1	9	4	9	9

Table 21: Pretest - Control Group

As anticipated, the mean value for correctness of the first and second questions tended to be significantly higher than those for the final two questions. Question one exhibited the highest mean (3.46 out of 5), and question three was the lowest (1.63 out of 5). Clarity in solution description for the first two questions tended to be much higher as well,

which reflected the accuracy of the solution, since a correct solution was likely easier to explain. However, it also reflected familiarity with this type of problem and the relative simplicity of the problem as compared to the other two. Question four illustrated the widest variation in correctness and clarity, largely due to the added necessity for the participant to translate and determine the problem boundaries. Also of interest was the slight dissimilarity between correctness and clarity on questions three and four. In question three, it made sense that clarity would be higher as the problem had inherently unclear boundaries and direction. Therefore, participants could easily explain what they thought should be done in a concise manner, but still miss large segments of a required solution. In the case of the fourth question, it was common for respondents to provide an answer that was either the product of a guess or calculations not recorded on paper. This resulted in lower clarity scores, but did not preclude a correct answer.

4.3.2.2 Posttest

The posttest differed from the pretest in that questions with no answer were included in the separate question tallies. These scores and statistics are shown below in Table 22. In this case, only the correctness of the solution was analyzed and no value was assigned to clarity in writing code. Trends again reflected anticipated results, with higher numbers in the first question and steadily declining mean scores as the questions became progressively harder. The distribution for the first question was very flat, with the occurrences for each score from two to five being very similar. The other questions showed curves that were closer to normal distributions, so the overall result for total scores approximated a normal distribution as well.

	<u>Q1</u>	<u>Q2</u>	<u>Q3</u>	<u>Q4</u>
Answers	24	24	24	24
Mean	3.25	1.88	1.42	0.92
Std Dev	1.48	1.39	1.06	0.88
Variance	2.20	1.94	1.12	0.78
Skew	-0.47	0.66	0.72	1.01
Counts				
5's	6	1	0	0
4's	6	3	1	0
3's	4	3	3	2
2's	5	5	5	2
1's	2	9	11	12
0's	1	3	4	8

Table 22: Posttest - Control Group

4.3.2.3 Qualitative Information

Qualitative information for both the pretest and the posttest were recorded as a part of this study. By necessity, data for the pretest were collected for every answer since the solution style was not dictated by the test. On the other hand, most answers for the posttest did not include a qualitative observation since only evidence of coding appeared on the majority of submissions.

	<u>Q1</u>	<u>Q2</u>	<u>Q3</u>	<u>Q4</u>
Pseudocode	13	7	2	0
Picture	0	0	0	3
Word	6	4	16	3
Math	0	4	0	1
scribbles	0	0	0	6
psu/word	3	4	1	0
pic/word	2	0	0	5
math/word	0	1	0	0
math/psu	0	3	0	0
pic/psu	0	0	0	0
None	0	0	0	3

Table 23: Pretest Observations - Control Group

For all pretest questions other than the third, subjects used a wide range of techniques to convey their solutions. A simple majority used a reasonably structured form of pseudocode to answer the first question

(thirteen; see Table 23). Three more individuals used words that approximated pseudocode. The second question showed a broader range of tendencies, with pseudocode again leading the way with seven instances. As expected, the third problem solution was generally presented in prose, while the final problem was most likely to exhibit picture representations to help with the explanation.

	Control	Treatment
noncode solution	5	0
additional pseudocode	1	0
additional picture	1	7
additional words	2	3

Table 24: Posttest Observations - Control

The majority of responses for the posttest included solutions that exclusively featured programming language code. Rather than report these instances by question, all instances were summarized for both the control and treatment groups in Table 24. The first row actually represented a negative observation. In this case, the respondent was unable to provide a programming code solution, so they attempted to provide a word-based solution to the problem. Obviously, this does not exhibit an ability to program, only to problem-solve. Tests with “noncode” solutions were not omitted. Persons with this sort of answer exhibited in some fashion on the test that they understood that answers were expected to be in a programming language; however, they were following instructions to show their process in attempting to find a coding solution. It was significant to observe that five instances of a “noncode” solution appeared in the control group.

All other observations reflected instances where additional materials evident on the posttest were provided in addition to a code solution. Four separate individuals provided additional materials to their coded solution,

with variations of pseudocode, words, and pictures being used. No participant used additional materials on more than one problem.

4.3.3 Treatment Group

4.3.3.1 Pretest

Testing scores and observations for the treatment group showed several interesting differences from the control group. As with the pretest scores for the control group, scores for the treatment group were lower for question three (see Tables 21 and 25). However, scores for question four were much higher in relation to the other marks. Correctness values for questions one and two were slightly lower for the treatment group than they were for the control group. On the other hand, the number of questions left completely blank was limited to two instances for the treatment group (once on question one and once on question four), which contrasted with nine instances for the control group. The net result, if zero scores were included in the average, would be closer between the two groups for question two and for the clarity scores for question one. However, since the sizable number of omissions for questions three in the control group emphasized a huge gap that might have reflected time constraints more than ability, zeros were omitted in the mean scores.

	<u>Q1</u>	<u>Q1</u>	<u>Q2</u>	<u>Q2</u>	<u>Q3</u>	<u>Q3</u>	<u>Q4</u>	<u>Q4</u>
	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>	<u>Correct</u>	<u>Clarity</u>
Answers	14	14	15	15	15	15	14	14
Mean	2.93	3.64	2.93	2.93	1.73	2.13	2.57	2.57
Std Dev	1.33	0.63	1.03	0.96	0.96	0.83	1.60	1.65
Variance	1.76	0.40	1.07	0.92	0.92	0.70	2.57	2.73
Skew	-0.08	-1.69	-0.30	-0.41	1.17	0.58	-0.09	-0.15
Counts								
5's	2	0	0	0	0	0	1	2
4's	2	10	6	5	1	1	5	2
3's	6	3	3	5	2	3	1	4
2's	1	1	5	4	4	8	2	2
1's	3	0	1	1	8	3	4	2

Table 25: Pretest - Treatment Group

As with the control group, the distributions for most question scores exhibited normal distributions. However, correctness values for question two exhibited a somewhat bimodal distribution, as did the correctness values for question four. Question four values tended to show a flatter curve of distribution, like the control group. Once again, the total of all scores maintained a normal distribution; hence, the means and variances for total score data provided useful information for later analysis. Total score averages for correctness were calculated at 9.80 for the treatment group and 9.68 for the control, which represented a small and likely insignificant difference. On the other hand, clarity scores for the treatment group averaged a total of 10.87, compared to 9.75 for the control group. This difference was sizable enough to analyze for statistical significance.

4.3.3.2 Posttest

	<u>Q1</u>	<u>Q2</u>	<u>Q3</u>	<u>Q4</u>
Answers	15	15	15	15
Mean	3.73	2.53	1.73	1.60
Std Dev	1.62	1.55	1.22	1.30
Variance	2.64	2.41	1.50	1.69
Skew	-0.89	0.26	0.59	0.20
Counts				
5's	8	2	0	0
4's	1	3	2	1
3's	3	1	1	3
2's	0	5	5	4
1's	3	3	5	3
0's	0	1	2	4

Table 26: Posttest - Treatment Group

Unlike the pretest, posttest scores of zero were recorded and added to the mean, since time constraints were known to be consistent for the posttest. As with the control group, scores were progressively lower for each question on the posttest, which again reflected the relative difficulty of each question. Distributions also showed similar characteristics to the control group. In both groups, there were cases where a division between students who 'understood' and those who did not were clearly visible. For example, question one in the treatment group (Table 26) showed that

most participants were able to achieve success, with the notable exception of a small subset who tended to do poorly throughout the posttest.

Although the pretest data showed possible differences in clarity results but none in correctness, the posttest scores exhibited more differences between the two groups. In fact, one might conclude that differences between the two groups at the beginning of the CS1 course, based on pretest correctness scores, were not significant. On the other hand, pretest scores for each question were higher for the treatment group than they were for the control group. Although each question score was not statistically significant on its own, the aggregate score mean (9.60 for the treatment group, compared to 7.46 for the control group) was of interest for analysis.

4.3.3.3 Qualitative Information

Observations for styles of presentation for the treatment group (see Table 27) showed stronger tendencies for particular approaches than the control group. The use of pseudocode made a much stronger appearance in the first two questions, and a second pass through the pretests revealed that the pseudocode structure was generally improved. The use of graphic helpers also increased in quality and clarity for the final question. Many of the control group produced illegible scribbles; however, the treatment group produced pictures, diagrams, and notes (which could be referenced multiple times as they attempted to solve the problem).

	Q1	Q2	Q3	Q4
pseudocode	11	7	0	0
picture	0	0	0	3
word	1	1	12	5
math	0	1	0	0
scribbles	0	0	0	0
psu/word	1	2	1	0
pic/word	0	0	2	3
math/word	0	1	0	1
math/psu	0	3	0	0
pic/psu	1	0	0	0
none	0	0	0	2

Table 27: Pretest Observations - Treatment Group

4.3.4 Outlier Groups

Most outlier groups consisted of one or two instances, which was insufficient to draw conclusions about the population they represented. These records may be viewed in Appendix XIV. However, it was instructive to view subsets of instances for those students who failed to complete the CS1 course, to determine if this information illustrated a pattern in pretest scores that was predictive of an individual's failure to complete the programming course.

4.3.4.1 Pretest

Between the two courses, there were sixty-two enrollees; two instances were removed due to missing informed consent forms and pretest data. Of the remaining sixty persons, forty-four completed the programming (CS1) course. Those who successfully completed the course averaged a correctness score of 10.09 and a clarity score of 10.50. On the other hand, those who failed to complete averaged 8.06 and 9.31, respectively. Individual question scores tended to be lower for each question with the exception of question three (the large domain problem), which was very nearly an exact match. The most dramatic differences were found in question four, which was the brainteaser problem. Those who completed had a 2.43 average, and those who did not had a 1.40 average for

correctness. When compared within specific classes, the correctness numbers for the fall were 2.27 for those who completed and 1.14 for those who did not. Similarly, correctness numbers for the spring were 2.61 and 1.62. Question one and question two scores for the fall semester group were fairly close, regardless of completion status. Question two scores were actually slightly higher for those who did not complete in the fall. Numbers in the spring show that those who failed to complete had slightly lower scores in questions one and two, with no real difference in question three. Therefore, problems such as the one found in question four may have some predictive capability for future success in a programming course.

4.3.4.2 Posttest

Five instances existed for students who completed the CS1 course but did not qualify for either the control or treatment group. However, these outlier records fell into three different outlier groups, so little value was expected to result from working with this data. Complete data recordings can be found in Appendix XIV.

4.3.4.3 Qualitative Information

Students who failed to complete the CS1 course tended to use illegible scribbles or no description at all in support of their efforts to solve the fourth question. Well-structured pseudocode was slightly less common, although still used strongly by the three students who took the treatment CS0 course but did not complete the CS1 course (Spring 5 group). Otherwise, there were no verifiable trends that revealed a difference between those who completed and those who did not complete the programming course.

4.4 Self-Evaluation Data

All self-evaluation data consisted of rankings of comfort based on whole values from one to five, with five being the highest level of comfort in one's ability to perform. The programming rating was an exception in that it included a value of zero for those persons who had no programming experience prior to the CS1 course. The addition of the zero rating provided a more accurate picture of prior experience, both in the averages and in the pure numbers. Otherwise, the growth of persons starting with no programming experience would have been lost in the data.

4.4.1 Control Group

The pretest data consisted of entries from twenty-two of the twenty-four control group members. The two exceptions included persons who chose not to answer the entire set of self-evaluation questions. They did, however, answer the questions at the end of the course in the exit survey. Their data was included in the aggregate post values, but their records were not included in paired analysis.

	<u>Computing</u>		<u>Math</u>		<u>Problem- solve</u>		<u>Programming</u>	
	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>
Answers	22	24	22	24	22	24	22	24
Mean	4.00	4.25	3.91	4.04	3.93	3.96	2.55	3.71
Std Dev	0.62	0.94	0.75	0.75	0.73	0.69	1.84	0.95
Variance	0.38	0.89	0.56	0.56	0.53	0.48	3.40	0.91
Skew	0.00	-0.89	0.15	-0.07	0.16	0.05	-0.35	-0.99
<i>Counts</i>								
5's	4	13	5	7	5	5	3	4
4's	14	5	10	11	10	13	5	12
3's	4	5	7	6	6	6	6	6
2's	0	1	0	0	0	0	1	1
1's	0	0	0	0	0	0	1	1
0's	0	0	0	0	0	0	6	0

Table 28: Self-Evaluation - Control

Computing

The computing score was intended to measure general comfort in working with computers. It was expected that these ratings would be above average, and both the pre and post means (see Table 28) supported this expectation. Although the change in the mean was not necessarily significant, it was important to note that a sizable number of participants migrated from rating themselves with a four (very comfortable) to a five (extremely comfortable). One of the two missing subjects rated him or herself with a 'three' for this data point, while the other chose a 'five'. This growth in confidence was viewed as a reflection of students' confidence in their ability to continue in computer science.

Math

The math score was expected to be average or above, and marks for the control group met this expectation. There was a very slight increase in the mean over the term, but this might have reflected the addition of the two entries that failed to submit initial data. Also, it was not possible to ascertain what outside influence may have altered participant self-ratings when there was, in fact, a change. However, it was encouraging to note that these numbers did not change significantly, which indicated that this group had some internal consistency in rating their comfort levels.

Problem-solving

As with math, problem-solving comfort levels did not change markedly for the participants, which also indicated that intra-rater reliability is high for this particular group of participants. Thus, the confidence level for changes in ratings for the programming characteristic was higher. Scores were similar to those exhibited in the math question and were generally above average.

Programming

Scores for programming showed a significant change, which was expected since six participants indicated that they had no programming experience prior to the CS1 course. In general, those who had

programmed before reported average or better comfort with writing program code. This trend continued with the posttest; those who had not coded before joined those with prior experience, reporting average or better rankings of self-confidence.

4.4.2 Treatment Group

	<u>Computing</u>		<u>Math</u>		<u>Problem-solve</u>		<u>Programming</u>	
	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>	<u>Pre</u>	<u>Post</u>
Answers	15	15	15	15	15	15	15	15
Mean	4.13	4.53	4.00	4.07	3.47	3.93	1.80	3.67
Std Dev	1.06	0.74	0.65	0.80	1.06	0.80	2.18	1.11
Variance	1.12	0.55	0.43	0.64	1.12	0.64	4.74	1.24
Skew	-1.96	-1.33	0.00	-0.13	-1.15	-0.84	0.49	-1.02
<i>Counts</i>								
5's	6	10	3	5	1	3	2	3
4's	7	3	9	6	9	9	4	7
3's	1	2	3	4	2	2	0	3
2's	0	0	0	0	2	1	0	1
1's	1	0	0	0	1	0	1	1
0's	0	0	0	0	0	0	8	0

Table 29: Self-Evaluation - Treatment

Computing

Computing scores were similar to those exhibited by the control group, with a single outlier ranking him or herself as being very uncomfortable with computing. This same individual indicated a low comfort level for problem-solving and no programming experience. Otherwise, participants tended to shift upward in the posttest, just as they had in the control group. Means were slightly higher for both the pre and post values for the treatment group.

Math

Math scores were nearly identical between the treatment and control groups. Similarly, the scores were nearly identical between the beginning and the end of the CS1 course. This indicated that intra-rater reliability for

confidence self-rating was sufficiently high to consider the programming rankings to be reasonably accurate.

Problem-solving

A mild anomaly in the problem-solving data can be seen in Table 29. Three individuals ranked themselves with below average comfort in problem-solving, which resulted in a difference in the mean between the control and treatment groups of 3.47 (treatment) to 3.93 (control). The exit survey scores were extremely close (3.96 control and 3.93 treatment). The starting values were not statistically significant, but the difference in values could be attributed to any number of factors. One possible explanation was that these participants rated themselves harshly as they remembered some of the activities and exercises provided by the new curriculum in the CS0 course. If this were the case, the change from pre to post was an indication that they had overcome their discomfort.

Programming

Over half of the participants reported no prior experience with programming (eight persons) as opposed to one quarter (six of twenty-four) of the control group; therefore, members of the treatment group had far less experience to which they could refer during the CS1 course. With zero values included in the calculation, the treatment group had 1.80 mean, compared to a 2.55 mean for the control group. However, without zero values, those with prior programming experience had a mean of 3.86 (seven persons) for the treatment group, compared to 3.50 for the control group (sixteen persons). Values reported at the end of the CS1 course were quite similar for both groups, despite the larger number of inexperienced members in the treatment group (3.71 to 3.67). However, upon the removal of persons with prior programming experience, members of the control group averaged a comfort score of 3.0, while a similar subset in the treatment group averaged 3.25. Starting values for these persons were zero, so these numbers also represented their growth in comfort. Growth exhibited by the treatment group was greater (+2.88

control versus +3.22 treatment) when those who identified themselves as having low vales at the beginning of the CS1 course were included..

There were single instances of persons reporting a reduced comfort level with programming in each group; in these instances, the reduction was from four to three or five to four. In general, those rating themselves with a three or higher at the beginning of CS1 tended to report little or no change in their comfort level. The treatment group averaged a final rating of 4.17 and the control a rating of 4.00, as opposed to starting marks for these persons that averaged 4.34 and 3.79 respectively. Therefore, the most interesting data was found in relation to those who identified no prior experience or those who reported discomfort with programming at the beginning of the class.

4.4.3 Outlier Instances

4.4.3.1 Outliers Failing to Complete CS1

The chart below summarizes the differences in self-ratings exhibited by individuals who completed or who failed to complete the programming course. Most values were tightly clustered. Self-ratings for comfort with math were consistently lower for those who failed to complete the course, although the differences (-0.53 control and -0.77 treatment) were not overly dramatic given the variation exhibited within the data. Interestingly, the control group reported more experience in programming than the treatment group, and no difference between those who succeeded or failed could be detected. The control group exhibited a difference in problem-solving; the treatment group did not.

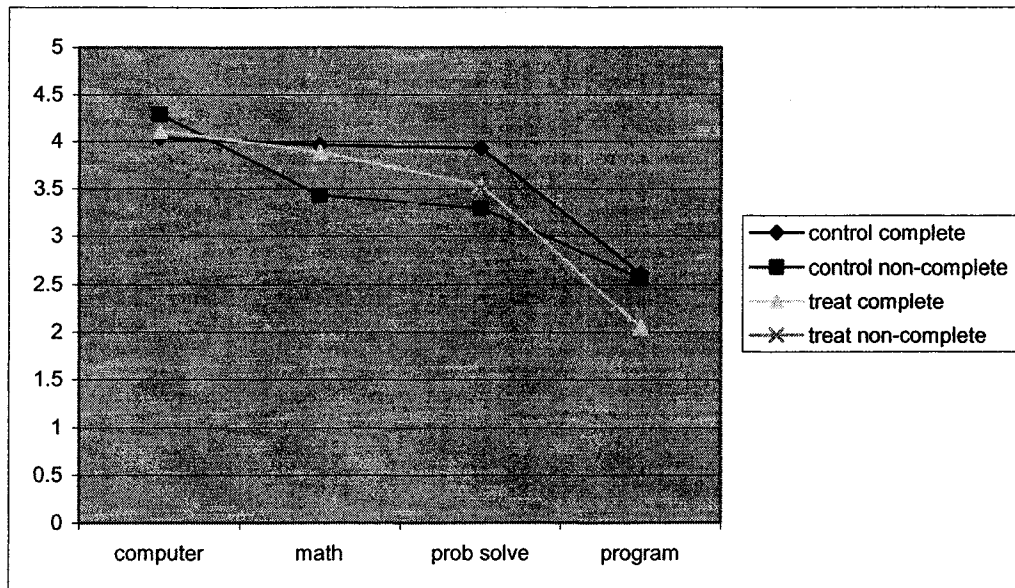


Figure 1: Completed vs Non-Completed Ratings

These comparisons provided an opportunity to isolate an external variable that might predict success in programming courses. Further study could isolate both mathematics skills and problem-solving skills more specifically, which would confirm work already completed by Quade (2003) and Stein (2002). However, this data did not isolate persons who took the treatment CS0 course. There were only three instances of persons failing to complete the course who had taken the CS0 treatment class, so the mean for such a small sample was not sufficient to discuss extensively. However, it was noted that these respondents did tend to rate themselves lower in math and problem-solving skills.

4.4.3.2 Outliers Completing CS1

Of those students who completed CS1 but did not qualify for the control or treatment group, all reported prior programming experience (although one reported very low comfort). There were only five instances that fell into this classification; thus, the mean score for programming was generally much higher than other subgroups. However, their marks were generally consistent with other individuals who completed the programming course.

4.5 CS0 Satisfaction Data

Participants were asked whether they felt the CS0 course was useful for their success in the subsequent CS1 course. This measure of satisfaction provided an indicator for student perceptions of the course's value. Higher satisfaction levels indicated that students were more convinced that they were learning useful things, so they were also more likely to be successful learning that which they needed to know.

	Control Pre	Control Post	Treat Pre	Treat Post
count	22	20	15	15
mean	3.32	2.95	4.40	4.00
stdev	0.99	1.28	0.63	1.20
var	0.99	1.63	0.40	1.43
skew	-0.08	-0.24	-0.55	-1.45

Table 30: Satisfaction Data

The ratings for the CS0 course showed a sizable difference between the control and treatment groups, especially given the five-point Likert scale used in this measurement. Scores remained a full point higher for those who attended the treatment course at both the beginning and the end of the CS1 course. Both groups exhibited some decline in satisfaction, but the chart below shows a similar slope for each. This decline can be attributed to the additional information known by the subjects at the conclusion of the CS1 course, which provided them with a better framework from which they could critique the CS0 course. At the beginning of the course, it was unlikely that the student was fully aware of how material in the prior course would relate to the new course. The evaluation at the end of the programming course provided a more informed analysis. This data appeared to be statistically significant and was analyzed further (see Chapter 5).

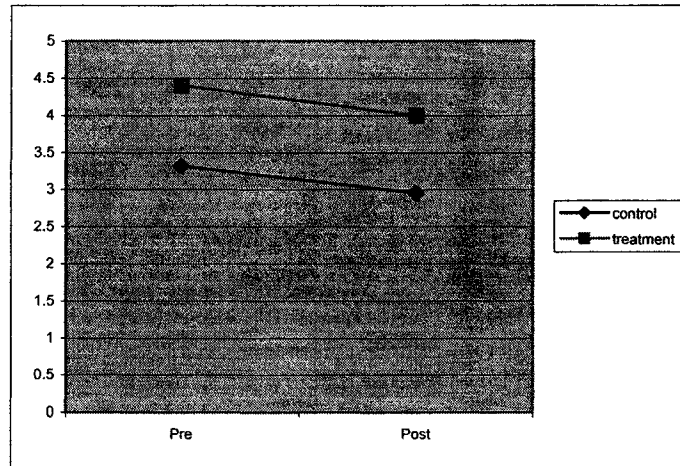


Figure 2: Change in Satisfaction Data

4.6 Qualitative Data

Qualitative data was collected in the demographics data collection tool and the exit survey in order to provide participants with an open forum for disclosing their thoughts on the CS0 course. In general, students kept their commentary very brief, if they gave it at all. However, indications were that the majority of participants considered their answers seriously and that they were hopeful that the information collected might be used to encourage improvement in the curriculum for the computer science program.

4.6.1 Best and Worst Parts of CS0

For both the control group and the treatment group, a majority of students identified some combination of problem-solving, algorithms, and thinking for programming as one of the most useful parts of the CS0 course. For the demographics data collection tool, which provided a view of opinions held at the beginning of the CS1 course, eight of fifteen in the treatment group and twelve of twenty-four in the control group identified problem-solving and algorithms as the most useful part of CS0. The ending view was provided by the exit survey. Results there showed dramatic increases, with all fifteen members of the treatment group citing these as

being most useful and seventeen of twenty-four of the control group doing the same.

Most Useful Comments from the Control Group:

“90% of the material was good.”

“study of algorithms most useful”

“how to write an algorithm”

“The most useful was the developing of algorithms. The in depth detail the instructor went into.”

“Helps me in using critical thinking to solve a problem”

“Most useful was the learning to analyze problems in order to solve them.”

Most Useful Comments from Treatment Group:

“...the instructor, his teaching style and testing style.”

“I liked all parts so I guess every bit of it was useful.”

“Algorithm development part and machine language was really helpful and I enjoyed it.”

“Practice with problem-solving. An overall view of how programming, problem-solving and computers are linked.”

“(CS0) put me in the mindset I needed to problem-solve in (CS1)...”

“The ways to approach a problem and working with algorithms.”

On the other hand, subjects had diverse views concerning what the least useful portions of the CS0 course were. Some of these comments included qualifiers that the student could understand why the subject material was included, indicating acknowledgement that not everything had to have apparent immediate application to have potential value. However, some areas were mentioned with some frequency. A few students in each group cited historical and social aspects of computing as having little use or value. Others cited exposure to assembly language

concepts, circuits, and binary numbers as negatives. Perhaps the most interesting commentary cited the very brief views of various programming languages provided in the existing CS0 course as being unproductive, with nearly a dozen separate instances of comments relating to this practice. In comparison, only six positive comments on this subject were found; all occurred in the demographic tool, with none occurring in the exit survey.

Least Useful Comments – Control Group

“least useful – The little bit of C++ that we used was vague and I felt uninformed.”

“The book did a poor job of explaining concepts”

“too much work”

“least useful was the design of circuits and the long boring lectures.”

“bit description and representation of values.”

“Learning fragments of programming languages. I remember asking (a question) and his answer made no sense.”

Least Useful Comments – Treatment Group

“history of computer (but need)”

“The Big-O notation part looked to me really frustrating.”

“learning about group work was not really put into practice.”

“dealing with assembly language, it was brief enough not to kill me, but annoying.”

4.6.2 Suggested Improvements

Very few comments were submitted regarding improvements; comments were often directed to both courses of study rather than being specific to CS0 or CS1. Those recorded that had a direct relationship to the CS0 course follow.

Control Group:

“Going over ideas in class is important, but lab time is essential in the quest to full understanding.”

“Use real life scenarios. Don’t use big words and work through problems using a structured walkthrough (step by step).”

“More problem-solving exercises...”

“Should teach more problem-solving techniques.”

“I really don’t feel that (CS0) helped tremendously with (CS1). Students should get started with actual programming ASAP. Pencil and paper algorithms just aren’t the same.”

Treatment Group:

“(No change to CS0) we covered a lot of information...the problem sets and tests were challenging, but that was a good thing.”

“Writing a paper about ethical issues – 2 class periods of class discussion would have been more thought provoking.”

4.6.3 Other Qualitative Data

The demographics data collection tool included a question asking subjects what they were most looking forward to and most worried about for the upcoming CS1 course. As expected, there were numerous comments about learning a programming language being what they were most looking forward to, and sometimes the thing they were most worried about. Students were primarily worried about grades and the amount of time and effort required for the course. Thus, no underlying themes were discovered that impacted the results of this study.

Control Group – most looking forward to:

“Program with quality instead of just programming”

“An excellent knowledge of C++”

“Programming with C++. This is what I’ve been looking forward to since last year.”

“Learning a programming language, gaining tools for understanding real-life tools in the ‘computer world’.”

Treatment Group – most looking forward:

“programming – I love to program.”

“I would like to learn as much as possible about C++. To be able to make programs.”

Control Group – most worried about:

“Grades and work demand. I am afraid of being short on time for completing the lab with understanding.” (emphasis included)

“worried about not being able to program correctly. That my code won’t work and I can’t figure out the solution.”

“Not being able to understand what we are working on. Hopefully that won’t happen.”

“May not be able to remember all the functions and syntax in C++”

Treatment Group – most worried about:

“Not being able to catch up with the rest of the class and not understanding the essence of given problems.”

“I’ve heard that this class is really tough and that the programs are really hard and time consuming. I’m worried that I’m not going to understand everything.”

5 Interpretations and Analysis

5.1 Sample Representation of Population

The sample selected for this study followed a cohort design, where persons within the target population were placed into their groups based on time period of attendance. Persons who entered the BSU computing course sequence for the control group were tracked during the Fall 2000 term in their CS1 course. A second cohort comprised the treatment group, which was measured during the Spring 2001 term in their CS1 section. In effect, this time-based selection effectively included all persons at BSU that fit the target population (persons entering computing studies at the introductory level) during the 2000/2001 academic year. There were no instances reported of individuals refusing to participate in this study.

However, simple inclusion of all members available that meet certain criteria at a given time did not mean that the sample was representative of the total population. It was important to determine whether the sample reflected the norm for the BSU overall computer science student population and the general student population during the academic year in question. According to public records provided by BSU's online data book, there were a total of 142 declared Computer Science (CS) and Computer and Information Sciences (CIS) majors during the target academic year (Bemidji State University, 2003). This same source reported an additional twelve CS minors. A simple projection indicated that approximately 40 persons enter the course of study per year and declare the major. Between the Fall 2000 and Spring 2001 terms, forty-four persons completed the CS1 course; thirty-nine were members of the control or treatment groups for this study. This indicated that this year was not atypical for enrollment in computing studies.

The institutional data book also reported that approximately nineteen to twenty-three CS and CIS majors are awarded Bachelor's degrees each year (not including minors). This seemed contrary to the projected forty students per class. However, it was necessary to also consider the reported six-year graduation rate, which was 47% for the years 1995-2001. Forty-seven percent (47%) of the forty-four students would produce twenty-one graduates from this group of students. Therefore, the sample was representative of a typical cohort for computing studies at BSU, given available data.

Student demographics at BSU are driven, to a large extent, by the demographics of the state of Minnesota, since 91% of all enrolled students are from in-state locations. Unfortunately, the race and ethnicity breakdown provided by the school included a large population of 'unknown' origin (33.25%), thus it was difficult to be certain about comparisons between general population and the sample population. However, the largest minority population was Native American (3.5% of the total population) followed by Asian descent, African descent and Hispanics (0.5% of the population each). In addition, there were international students that increased the Asian percentage, in part due to Malaysian and Japanese students, who accounted for 1% of the student population. Since the vast majority of enrolled students were from Minnesota, it was possible to extrapolate from census data provided by the Minnesota Department of Administration (2003). This resource reported that 90% of Minnesotans are 'white' or 'Caucasian' in race, thus it was reasonable to assume at least 82% (90% of 91% from Minnesota) of the students at BSU were Caucasian. In this study, 77% of those who reported their ethnicity were Caucasian, 19% reported Asian descent, and 3.8% reported that they were Native American. These numbers seemed reasonable, although the Asian population was inordinately high given the general BSU demographics. However, the Taulbee survey indicated that

21.7% of Bachelor's degrees in computer science awarded in the United States were given to students of Asian or Pacific Islander descent in 2001/2002 (Vardi, Finin & Henderson, 2003); those classified as 'white' or 'Caucasian' were awarded 57.8% of these degrees. This illustrated a tendency for the Asian population to be larger in computing studies, so this seeming disparity was normal for the field.

The Taulbee survey reported that the number of women receiving bachelor's degrees in computer science as being 18.8% of all awarded degrees, regardless of race or ethnicity (Vardi, *et al.*, 2003). No general data about gender was found in the BSU data book, but it was considered likely that campus-wide data would not show any consistency with the population in computer science. The sample included fifteen women. Twelve completed the CS1 course, representing 26.7% of the enrollees for both sections of the CS1 class. This was higher than nation-wide statistics, but it was consistent with national trends, which show much higher numbers of men than women in the computing disciplines.

Data with respect to age, disability and other factors were either not available for BSU, or were presented in ways that precluded a reasonable comparison. However, the large number of persons reporting ages between eighteen and twenty-two (forty of all persons in the sample) supported the contention that most persons in the program were within the traditional age span for a four-year program. Also, the inclusion of six-year graduation success rates indicated that it was not uncommon for BSU students to take more than four years to graduate, which helped to explain the higher number of sophomore and junior students in these classes. Four percent of the BSU population had recorded disabilities, whereas 3.8% of the sample reported a disability, which was also a reasonable match. However, since data collected in this study did not ask

if the disability was on record with the university, the comparison was not entirely valid.

The sample selected appeared to be a representative sample for the population of computing students at Bemidji State University. In cases where participant characteristics seemed divergent from the general BSU population, the difference was easily explained by national tendencies for computing degree programs. For example, the number of Asian participants and the number of males in computer science programs are typically higher. Similarly, differences in the sample from national tendencies were explained by the local demographics exhibited at BSU. For example, the national numbers indicate that only 0.4% of degree recipients are Native American, yet this population is a larger portion of the overall BSU enrollment than most campuses. Also, the number of minorities on campus was much lower at BSU than most metropolitan area schools. In summary, the sample was appropriate for the combination of school, location and discipline; therefore it was a representative sample.

5.2 Analysis of Programming Skill Learning

The posttest scores measured the ability of subjects to perform program coding tasks. The pretest provided a baseline ability marking in problem-solving skills in order to account for variance in ability between individuals. This pretest was given a correctness score and a clarity score. Thus, a question could receive high marks for a correct answer, yet low marks for poor presentation. On the other hand, an individual could receive high marks for presentation despite a poor answer. The posttest was marked solely on program code correctness.

The strategy for analyzing the data was to first determine if there was a difference for pretest correctness, pretest style and posttest marks between the control and treatment groups in order to determine whether there were any significant scoring differences between groups. Pretest correctness scores were then subtracted from posttest scores in order to determine a relative measure of change for each participant. These tests were not an exact measure of the same skill, so it was not important whether the scores were statistically different, nor did it matter whether they were higher or lower than previous scores. The difference represented the relative value based on the potential shown by the subject in the pretest as compared to the achievement exhibited in the posttest. Thus, analysis was performed across groups to ascertain the difference between potential achievement and actual achievement.

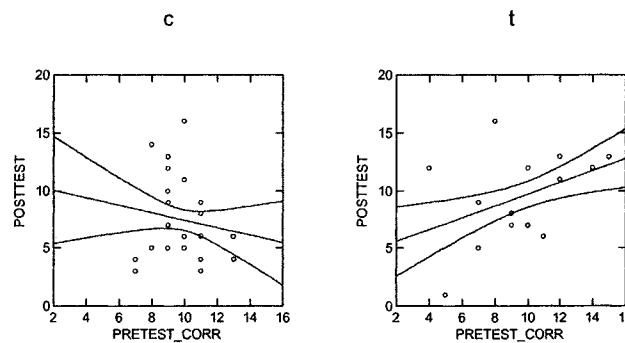


Figure 4: Pretest Correctness versus Posttest Scores

The first indication that there was something interesting happening with the test data can be seen in the graphs shown above with their best-fit lines. These charts show a 0.50 confidence interval on the regression line. The confidence interval supported the idea that the regression line was a good fit for the data collected, although it is possible that the control group could vary to a line with little or no slope. The control group is shown on the left and the treatment group on the right. This scatter plot shows the distribution of pretest correctness score versus the posttest score received by each subject. It was natural to expect that participants

who scored higher on the pretest would also score higher on the posttest. The data for the treatment group matched this expectation, but the data for the control group did not. Surprisingly, the control group participants tended to score lower on the programming posttest if they scored higher on the pretest.

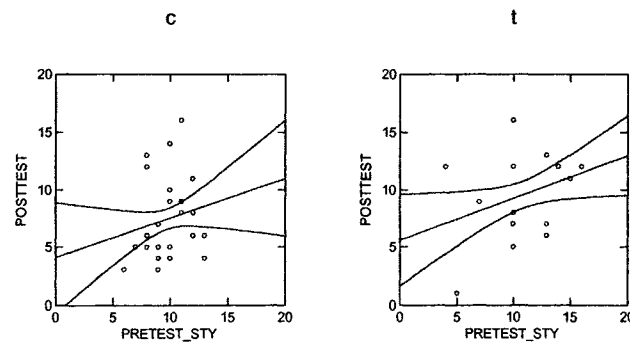


Figure 5: Pretest Style versus Posttest Scores

In order to obtain a better picture of the relationship between pretest and posttest scores, two more scatter plots with best-fit lines illustrate pretest style scores versus posttest scores. Again, the control group is on the left and the treatment group is on the right. Both best-fit lines indicate that higher style marks led to higher posttest scores. However, since the control group's regression line has little data on the extremes to confirm the slope of the line, it could potentially fall anywhere from a slightly downward slope to a more dramatic upward slope. Both scatter plots support the contention that higher style scores tended to also be met with higher posttest scores.

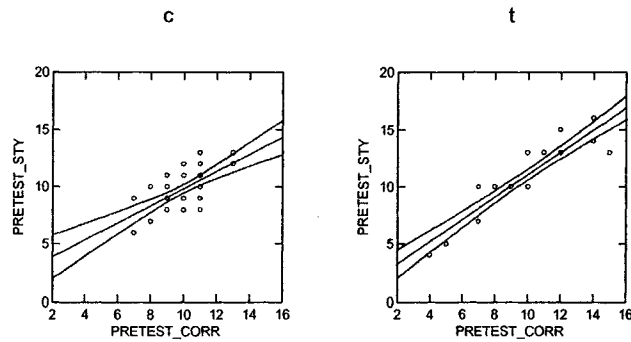


Figure 6: Pretest Correctness versus Pretest Style

Since it was possible for a person to score well on correctness and poorly on clarity/style, a scatter plot was created for each group to visualize correctness versus style scores. These plots are shown above with a regression line that has a 0.50 confidence interval. In both cases, participants scored equally well in both categories and the confidence interval indicates that this tendency was strong. This was not surprising, a clearly explained solution made it easier to ascertain if a solution was correct.

These three sets of plots indicated that the pretest correctness scores were not necessarily a good predictor of success in programming skills. On the other hand, the ability to clearly express a solution may be a better indicator for success. It was also possible that the inverse relationship shown in the control group may be a reflection of the need for tools to organize solutions into a formal style. Persons with excellent problem-solving skills certainly tend to make good programmers. However, problem-solving skills that cannot be organized and formalized do not help with programming. This is an excellent argument for the increased coverage of pseudocode, diagramming and problem-solving found in the treatment curriculum, since they all exercise the ability to structure and formalize problem-solving tasks.

Source	Sum-of-Squares	df	Mean-Square	F-ratio	P
Control vs Treatment	42.339	1	42.339	3.050	0.089
Error	513.558	37	13.880		

Table 31: Posttest Analysis of Variance

Posttest data were analyzed using an ANOVA with a constructed independent variable called "Group," which held either a 't' for the treatment group or a 'c' for the control group. An F-ratio was computed, along with its corresponding p-value in order to determine if the observed difference was statistically significant. This test had higher power than other tests and was appropriate given that the data to be tested was ratio data for repeated measures. Hence, this ANOVA served as the simple effects test for the repeated measure of the posttest across two groups, the control and the treatment group.

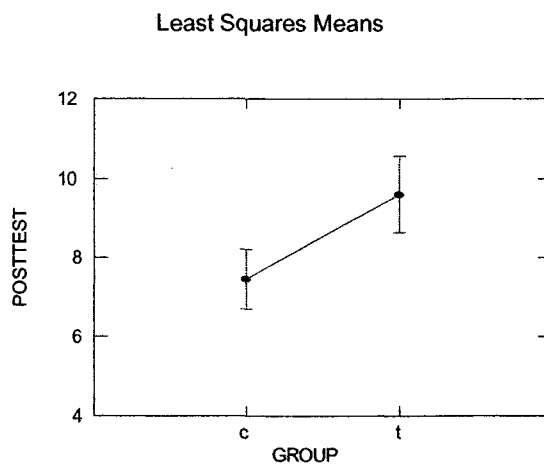


Figure 3: Posttest Across Control and Treatment

Visually, the least square means shows that the range for the control group and the treatment group did not overlap. This indicated that the difference in scores between the groups were a good candidate for

statistical significance. However, the data in Table 30 did not indicate a statistically significant difference. The p-value of 0.089 was within a confidence interval of 0.1. However, a confidence interval of 0.05 is normally required. Therefore, we can only state that members of the treatment group tend to score higher on the posttest in the observed data.

Source	Sum-of-Squares	df	Mean-Square	F-ratio	P
Control vs Treatment	0.052	1	0.052	0.009	0.924
Error	207.025	37	5.595		

Table 32: Pretest Correctness Analysis of Variance

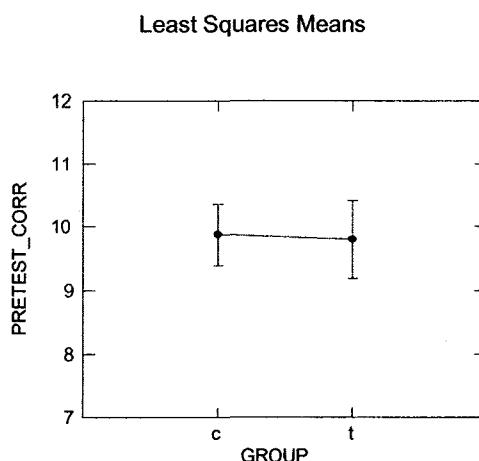


Figure 4: Pretest Correctness Across Groups

A similar analysis approach was applied to each pretest score, and the correctness values exhibited no difference. In fact, they showed a strong tendency to fall within the same range, regardless of whether an individual was in the control or treatment group. This result actually strengthened the significance of posttest results. Since participants apparently began the CS1 course with similar problem-solving ability for correctness, then one group did not enter with an advantage in raw skill. This conclusion

effectively eliminated that external variable and further isolated the treatment curriculum.

Source	Sum-of-Squares	df	Mean-Square	F-ratio	P
Control vs Treatment	11.510	1	11.510	1.688	0.202
Error	252.233	37	6.817		

Table 33: Pretest Style Analysis of Variance

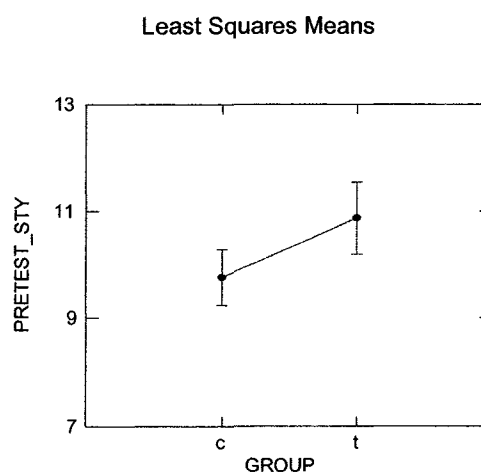


Figure 5: Pretest Clarity Across Groups

An ANOVA was also applied to the pretest clarity (or style) numbers to determine if the results were statistically different between groups. There did appear to be some visual difference between the two groups when a least square means plot was developed. However, there was clearly some overlap in the range between groups, which indicated the observed difference would not be statistically significant. The ANOVA values confirmed this interpretation and a p-value of 0.202 was calculated. Although this value indicated an observed difference, it was clearly outside of the 0.05 threshold value.

Once again, these results confirmed that the groups began the CS1 class with a comparable set of raw skills. There was some observable difference in the ability of participants to clearly outline solutions. This difference could translate into improved ability to program, but the lack of statistical significance here or in the posttest scores made it impossible to draw this conclusion.

Source	Sum-of-Squares	df	Mean-Square	F-ratio	P
Control vs Treatment	45.356	1	45.356	2.787	0.103
Error	602.233	37	16.277		

Table 34: Score Difference ANOVA

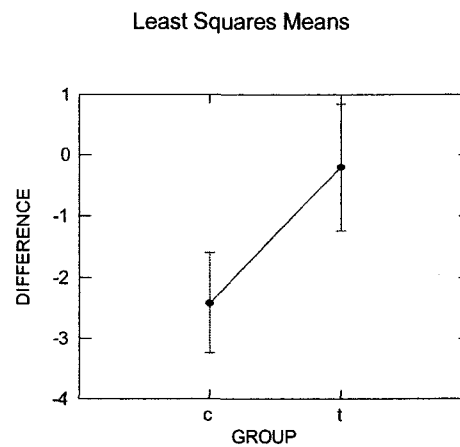


Figure 6: Difference in Scores Accross Groups

The concluding piece of statistical analysis on the test scores utilized the difference between the pretest correctness scores and the posttest scores. No analysis of the clarity scores versus posttest scores was undertaken since the posttest was simply a measure of program code correctness, not style. Although there was a visible and observed difference in the measures rendered for the two groups, ANOVA results again failed to render a p-value below the 0.05 threshold. The calculated

value was 0.103, therefore the observed difference cannot be considered statistically significant.

5.3 Analysis of Self-Evaluation of Skills

Participants were asked to rate their comfort in their ability to apply four different skills related to computer science. All of these characteristics were measured at the beginning and the end of the CS1 course in order to isolate any changes in confidence. Since two separate groups were measured at two different points in time, data were checked for significant differences in two ways. The first was to analyze possible changes over time, and the second was to analyze values between groups.

The analysis of rating changes over time was performed on each of the four characteristics for the two groups. This resulted in eight sets of calculations and constituted an attempt to determine if participants had altered their opinions for these characteristics after completion of the CS1 course. Collected data consisted of ordinal ratings from one to five, so a Chi square analysis was performed on the data. Since most of the data exhibited a standard distribution, an analysis of variance was performed by calculating F-values and their corresponding p-value in a repeated measures analysis (pre and post values). Furthermore, a t-test was performed for two independent means for analysis across groups.

	Chi square	df	P value	df x	df y	F(x,y)	p value
treat computing	3.934	4	0.415	1	14	6.000	0.028
treat math	1.243	4	0.871	1	14	0.189	0.670
treat problem-solve	2.334	4	0.675	1	14	5.914	0.029
treat programming	13.018	5	0.023	1	14	13.618	0.002
control computing	9.260	4	0.055	1	21	2.392	0.137
control math	0.667	4	0.955	1	21	2.100	0.162
control problem-solve	0.115	4	0.998	1	21	0.417	0.525
control programming	8.482	5	0.132	1	21	13.096	0.0016

Table 35: Self-Evaluation Change Over Time

The preceding table reports calculations for both the Chi square tests and the F tests performed for each characteristic. Chi square tests showed four degrees of freedom for all but the programming rating, which included zero values. The change in degrees of freedom for y in the F test reflected the difference in size between groups. The normally accepted threshold to indicate statistical significance is $p = 0.05$; all values below this threshold are highlighted in bold, italic print.

The only comparison that yielded a statistically significant answer for both tests was the difference between ratings expressed for the treatment group's programming skills. The Chi Square p-value of 0.023 was well below the required threshold, as was the F test's p-value of 0.002. However, because Chi Square can account for the bimodal distribution exhibited by these ratings and the analysis of variance could not, the first result was more important. The mere fact that both tests show statistical significance adds strength to the argument that there was a definite difference in comfort at the end compared to the beginning.

Other tests that illustrated significance using analysis of variance methods did not show the same consistency with the Chi square analysis. For example, values for the treatment group's problem-solving and computing scores were deemed to be significant, but were countered by high p-

values in the Chi square tests. This reflected distributions that exhibited higher skew values. For example, the skew for opening scores on computing was -1.96 and the corresponding skew at closing was -1.33 . This indicated that analysis of variance would not be wholly accurate. As Chi square does not rely on means and variance, it was not subject to the impact of this skew. Further, Chi square has lower power as a statistical test than the analysis of variance. If Chi square indicated that there was no significant difference for naturally ordinal data, the analysis of variance could not reliably state otherwise.

The only other characteristics that approached significance for the Chi-square test were those for the control group's computing and programming ratings. The p-value of 0.055 for the computing characteristic was barely above the threshold. The distribution of this data was close to normal with skew values of -0.89 at the beginning and 0.0 at the end. Thus, the analysis of variance had some value, but it returned only a p-value of 0.137, which was not statistically significant. In other words, some differences were observed in the data, but it did not meet sufficient criteria to overcome the likelihood that these results might have occurred by chance. Similarly, the values for the programming scores in the control group had a low enough p-value to suggest some difference in this data. However, the bimodal distribution again rendered the analysis of variance moot and the Chi square must be relied upon for the measure of statistical significance.

Perhaps the most important aspect of the analysis over time was that most characteristics very clearly showed no change between the beginning and ending measurements. In particular, the math and problem-solving ratings for the control group exhibited data that remained remarkably similar. Overall, the programming ratings were the only characteristic to show observable change for both groups. These results

verified the anticipated change in the subjects' comfort with their programming skills by showing consistent intra-rater marks for characteristics not anticipated to change during the time frame measured. Thus, the statistically significant difference shown by the treatment group in rating programming confidence has greater strength given the relative stability of other characteristics.

	pre computing	pre math	pre prob solve	pre programming	post computing	post math	post prob solve	post programming
Chi square	4.35	0.86	6.91	6.49	1.07	0.12	1.94	0.21
df	4	4	4	5	4	4	4	5
p value	0.360	0.930	0.140	0.261	0.899	0.998	0.746	0.999
df			35	35				
t test			1.57	1.13				
p value			.1<p<.2	.1<p<.2				

Table 36: Satisfaction Measurements Between Groups

The analysis between groups yielded no results within the threshold for statistical significance. The observed difference between the treatment and control groups in problem-solving ratings at the beginning of the CS1 course came closest to achieving this status. In this case, the treatment group scores were observed to be lower, but the Chi square p-value of 0.14 was not adequate to state that this result was more than a chance occurrence. An analysis of variance was run on this data, since it had a well-behaved set of variables, but no statistical significance in the difference was found.

There was also an observed difference between the skill comfort level ratings in programming at the beginning of the CS0 course. This was not deemed to be statistically significant by the Chi square test, with a p-value of 0.261. The analysis of variance for these scores meant little for bimodal data. The raw data (with means of 1.80 for the treatment group and 2.55

for the control group) encouraged belief that there was a significant difference between the groups. However, the double peaks in the distribution of scores seen in the chart below illustrate how mean values can mislead analysis.

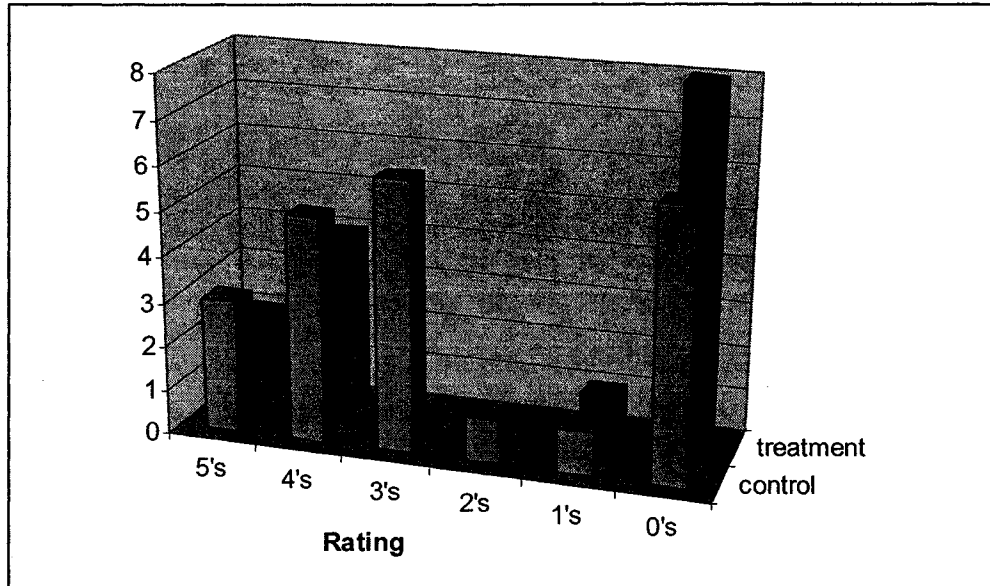


Figure 7: Bimodal Programming Rating

Again, the lack of variability across groups was as important as the observed differences. In particular, the ratings at the end of the CS1 course were extremely similar between the two groups. The math and programming ratings were nearly a statistical match, with strong similarities in the other two items. At the beginning of the CS1 course, only the math rating showed an extremely strong similarity. There were observable, but not statistically significant, differences in self-ratings for problem-solving and programming skills. These results both show the treatment group rating themselves lower than the control group at the beginning of the course, only to rate themselves as equals at the end of the course.

5.4 Analysis of CS0 Satisfaction Levels

The satisfaction raw data reported in Section 4.5 showed an observable difference between response data given by the control group compared to that given by the treatment group. The box plots shown below further illustrate the apparent difference between the treatment and control groups. The area of the box represents the most common response area for each group. There did not appear to be overlap on the scale between box plots for treatment versus control data during the same measurement period. Also, these plots showed no apparently significant difference within groups between the two measurement instances. There does appear to be significant overlap of the boxes in the plots between both treatment plots and both control plots.

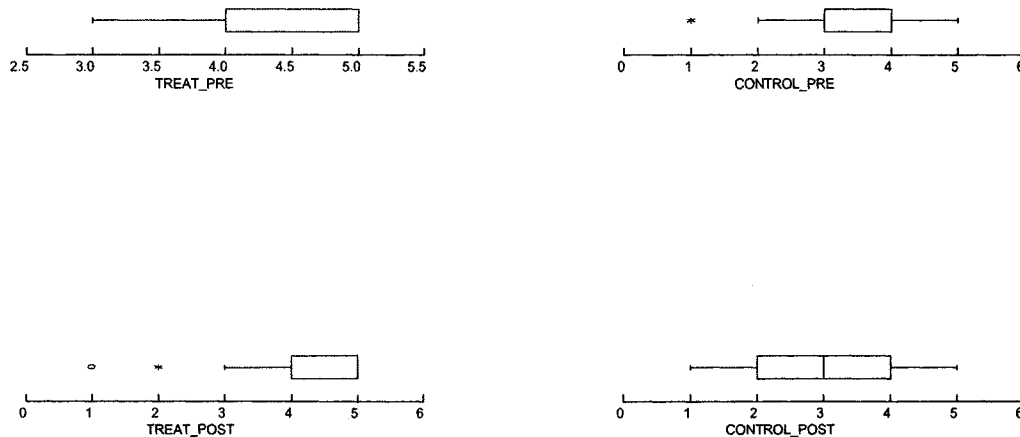


Figure 8: Box Plots for Pretest and Posttest

In order to test the difference between the treatment and the control groups, a Chi-square test was first used since the data was inherently ordinal in type. However, a t-test for two independent means was also used, since it was known that the data exhibited a fairly normal distribution for all but the treatment posttest, which exhibited two apparent outliers. The data qualified for the t-test because data was collected from an entire cohort for each group that consisted of all members of the sample

population for that period in time. The variable for partitioning was the term of attendance for CS1, with the qualifier that a pertinent CS0 course was attended prior to this course.

	Chi Square	df	pvalue
Beginning CS1	10.45	4	0.033
End CS1	7.55	4	0.110
Both Ratings	16.76	4	0.002

Table 37: Satisfaction - Chi Square Test

The results of Chi square analysis showed that when all ratings from both the pre and post measurements were used, there was a highly significant finding of difference between the control and treatment group. A p-value of 0.002 indicated that there was a two in one thousand chance that this difference was merely the result of a chance set of circumstances. Analysis between groups at the beginning and the end were undertaken with strong results at the beginning measurement. The difference between the two groups was again statistically significant with a p-value of 0.033. On the other hand, the two instances shown outside the box plot range limit the difference at the end of the CS1 course to an observable, but not statistically significant difference. A p-value of 0.110 indicated a strong possibility that a difference existed normally for the population.

	Mean Treatment	Mean Control	N Treatment	N Control	df	t value	p value
Begin CS1	4.40	3.32	15	22	35	3.72	0.001
End CS1	4.00	2.95	15	20	33	2.47	0.02

Table 38: Satisfaction - Independent Means

Two instances were tested by the t-test for two independent means; the results can be found in Table 38. The first set of measurements occurred at the beginning of the CS1 course and data was collected on student satisfaction with the prior CS0 course. The treatment group consisted of

fifteen subjects and a mean of 4.40 was calculated. The control group consisted of twenty-two records with a mean of 3.32. With thirty-five degrees of freedom, the t-value indicated an extremely high confidence level ($p=0.001$) that there was a significant difference between the CS0 satisfaction level of the control and treatment groups at the beginning of the CS1 course.

The second set of measurements was taken at the end of the CS1 course with a similar question and rating structure. The treatment group consisted of fifteen subjects compared to twenty for the control group (two additional members failed to answer this question). Mean values were slightly lower for each (4.00 for the treatment group and 2.95 for the control group). It must be noted that several of instances in the treatment group were not consistent within the data, providing a skew value of -1.45 . With thirty-three degrees of freedom, the t-value again indicated a high confidence level of $p=0.02$. Therefore, the difference between measurements at the end of the CS1 course for satisfaction levels between the control and treatment groups appeared to be statistically significant. However, the findings of the t-test were weaker than indicated by the p-value due to the results of the Chi-square analysis and the skewed distribution. Therefore, it was only possible to state that the difference at the beginning was statistically significant. The combined results of the Chi-square test indicated that there was likely a continued difference at the end of the CS1 course as well.

Although there was a significant difference in satisfaction between groups, observed data also showed a decline in ratings between the beginning and ending measurements. In order to test this for significance, an F-test (analysis of variance) was undertaken. The format of this study fits the repeated measures design (also known as the treatments by subjects design). Only persons who submitted ratings at both the beginning and

the end of the CS1 course were used for this analysis. If an individual chose not to submit a rating during one of the two measurements, their data were removed as this test looks for significance in changing values for participants.

Source	Sum of Squares	df	mean squares	F value	p value
Total	48.32	37			
Subjects	28.32	18			
Measures	0.95	1	0.947	0.895	0.357
Error	19.05	18	1.058		

Table 39: Control Satisfaction Analysis of Variance

The test for the control group yielded nineteen pairs of useful data. Although the control group consisted of twenty-four individuals, two individuals chose not to answer this question in the first measure and four did not answer in the second (one chose not to answer both times). Nineteen subjects over two measurements yielded a total of thirty-eight values for total of thirty-seven degrees of freedom. The interest lies in the possibility that the values across measurements differ. With two measurements, the degrees of freedom value for calculation of the F-value was one. Given the value for $F(1,18) = 0.895$, the p value was 0.357. This means that this F-value would occur by chance roughly 35% of the time, which is extremely high. Therefore there was no significant difference between the two measurements in the control group.

Source	Sum of Squares	df	mean squares	F value	p value
Total	26.80	29			
Subjects	19.80	14			
Measures	1.20	1	1.200	2.897	0.111
Error	5.80	14	0.414		

Table 40: Treatment Satisfaction Analysis of Variance

The treatment yielded fifteen pairs of useful data for analysis. Again, there were two measurements, thus the degrees of freedom were one for measurements and fourteen for the subjects. The $F(1,14)$ value was 2.897 which yielded a p-value of 0.111. Although it was quite a bit closer to indicating that some difference might be consistently found with additional measurements in the future, it showed no significant difference in this data.

5.5 Data Mining Analysis

5.5.1 Unsupervised Clustering Analysis

Unsupervised clustering works with a set of input attributes for all instances of a data set and attempts to find clusters of like instances. To accomplish this task with the given data, the group identifier field was removed so that it could not influence the grouping of instances. Data was placed into a format that allowed the use of the iData Analyzer software provided with the Roiger and Geatz (2003) text (for details of the data setup and data mining processes, please visit chapter three). All data mining sessions utilized the ESX tool provided by the software. Several sessions were undertaken with varying subsets of attributes used as input for the data. Three of the more interesting and/or representative sessions are discussed in this section.

5.5.1.1 Unsupervised Cluster with All Attributes

	Class 1	Class 2	Class 3	Class 4	Domain
Res. Score:	0.323	0.382	0.358	0.35	0.29
No. of Inst.	18	3	3	15	39
Cluster Quality:	0.11	0.32	0.23	0.21	

Table 41: Unsupervised Session 1 Clusters

The unsupervised clustering with all attributes as input fields resulted in four classes when an ESX similarity score of 35 was used and a tolerance of 1.0 for real valued data was selected. The overall domain resemblance score was 0.29. Thus, resemblance scores over this were expected for each class. A higher cluster quality number indicated how closely members of each class resembled each other. This session exhibited a pattern seen in most sessions with this data; two classes tended to claim a majority of the instances, while other classes included much smaller numbers of instances. In this case, thirty-three of the thirty-nine records belonged to either class 1 or class 4.

	Class 1	Class 2	Class 3	Class 4	Domain
pretot1 (mean)	9.39	7.00	11.33	10.67	9.85
(sd)	2.00	3.00	1.53	2.26	2.33
pretot2 (mean)	9.72	6.67	10.00	11.47	10.18
(sd)	2.42	2.52	2.65	2.26	2.64
posttot (mean)	5.94	8.67	4.33	11.80	8.28
(sd)	2.44	3.51	1.53	2.60	3.83
diff both (mean)	(3.61)	1.83	(6.33)	0.73	(1.73)
(sd)	2.32	6.25	3.06	3.29	3.96
Most common attribute values					
group	c	t	c	t	
excomputer	5	3	5	5	
exmath	4	5	3	5	
exprobsolve	4	2	3	4	
exprog	4	1	3	4	
colgrade	b	a	c	a	
Cs0 grade	b	b	c	a	
precomputer	4	5	4	4	
premath	4	4	3	4	
preprobsol	3	4	4	4	
preprog	4	0	0	0	

Table 42: Common Attributes Session One

The two larger groups (groups 1 and 4) could be characterized as those instances where the individual generally did poorer on the posttest (class 1) and those who did well on the posttest (class 4). In fact, these groups also tended to show some difference in pretest scores as well, with a slight advantage in pretest scores. However, the difference in scores between the pre and posttest showed a decided drop (an average drop of 3.61 out of 20) for the class 1 instances. Of interest is the fact that most of the class 1 members were from the control group (fourteen), with only a small number from the treatment group (four). On the other hand, the majority of class 4 members were from the treatment group (nine) as opposed to the control group (six). Both groups tended to rate themselves highly on their own skill levels, but self-reported grades differed, with a tendency toward A's in group 4 and B's in group 1 for both the CS0 course and college grades in general. On the other hand, most of group 4

members had no prior programming experience, while most of group 1 had such experience.

The smaller groups identified two special case groups within the data. Class 3 identified a set of three instances where individuals reported a history of poorer performance with C grades and lower self-assessment ratings. These individuals did well on the pretest and very poorly on the posttest, showing significant drops in their scores. All three members of this class came from the control group. On the other hand, class 2 consisted of two members from the treatment group and one from the control group. These individuals tended to do poorly on the pretest and roughly as well on the posttest. They tended to report better grades, but gave themselves lower self-assessment ratings at the end of the CS1 class.

Class 1	Class 2	Class 3	Class 4
Exprog =4	2 <= diff <= 2	-9 < diff <= -7	7 <= posttot <= 16
Post2 = 1		Colgrade = c	-3 <= diff <= 7
Post3 = 1			Colgrade = a
Post4 = 1			CS0 = a
10 <=pretot2 <= 13			
1 <= posttot <= 9			
-6.50 <= diff <= -1			
Colgrade = b			

Table 43: Session One Rules

The iData Analyzer software generated a set of rules for each of the four classes. These are displayed in the previous table and clearly indicate that a broader set of conditions applied to the two larger groups. Of particular note for class 3 is the large drop in scores (-9 to -7) between the pretest and posttest combined with a tendency to receive lower grades in college than most other participants in the study. While the pretest scores

indicated that these persons had raw ability in problem-solving, the self-reported average college grade indicated that these persons may be having a difficult time adapting to the college environment.

The differences between the two larger classes (class 1 and class 4) were of particular interest. It was clear that the posttest scores covered different ranges (1 to 9 in class 1 and 7 to 16 in class 2); thus, it seemed that a major part of the grouping was based on score success. Similarly, the difference between pretest and posttest scores tended to be more positive for class 4 than for class 1. This seemed to be a positive indicator that the treatment group had fared well, since the majority of instances in class 1 belong to the control group and the majority of class 2 to the treatment group. However, countering this was the tendency for members of class 4 to have self-reported A-grades in college, versus B-grades in class 1. Furthermore, class 4 instances also received more A grades in the CS0 course (although this seemed mildly redundant as participants reported an A average for all college courses). Therefore, part of the grouping was influenced by overall college performance, which could account for some of the uncertainty exhibited in the data analysis presented in prior sections.

5.5.1.2 Unsupervised Cluster with Grade History Removed

	Class 1	Class 2	Class 3	Class 4	Class 5
Res. Score:	0.355	0.397	0.353	0.365	0.353
No. of Inst.	18	4	2	13	2
Cluster Quality:	0.13	0.27	0.13	0.17	0.13

Table 44: Unsupervised Session 2 Clusters

The prior data mining session illustrated in 5.5.1.1 indicated that the college grade performance might have had a strong influence over the classification of individuals. Thus, another session was run with the grade history attributes for college, high school and CS0 removed to test their

impact. The same similarity scores and software variables were selected and the resulting classes were fairly similar with respect to the numbers of control and treatment instances in the two larger groups of individuals. Class 1 consisted of thirteen control group and five treatment group members, whereas there were eight treatment and five control members in class 4.

	Class 1	Class 2	Class 3	Class 4	Class 5	Domain
pretot1 (mean)	9.17	10.50	10.00	11.23	5.50	9.85
(sd)	1.89	1.73	1.41	2.20	2.12	2.33
pretot2 (mean)	9.39	9.75	8.50	12.39	5.50	10.18
(sd)	1.94	2.36	0.71	2.10	2.12	2.64
posttot (mean)	7.78	5.00	8.00	9.69	10.50	8.28
(sd)	4.25	0.82	7.07	3.01	2.12	3.83
diff both (mean)	(1.50)	(5.13)	(1.25)	(2.12)	5.00	(1.73)
(sd)	4.02	2.66	8.13	2.53	4.24	3.96

Table 45: Session Two Attributes

There was a shift of instances across groups once self-reported grades were removed; however, the core of most of the groups remained. Class 5 consisted of two instances that came from class 2 of the prior session. It became evident that these two individuals stood out because they rated themselves extremely poor at problem-solving and computing at the beginning of the CS1 course. Both were members of the treatment group and both improved markedly between the pretest and posttest. Class 2 consisted of individuals who showed a pronounced drop between the pretest and posttest and consisted of two instances from class 3 of the prior data mining session. Two additional control group instances joined this set, which clearly represented those who did comparatively poorly in the posttest as compared to the pretest indications. Class three presented an oddity, the individuals did equally well on the pretest, but one improved dramatically, while the other declined dramatically. Both members came from the control group. The pairing appeared to have originated due to an exact match of individual question scores on the pretest.

The two larger classes did not show the pronounced change in difference between pretest and posttest scores, yet they did show higher pretest and posttest scores for class 4 versus class 1. Again, class 4 is largely populated with treatment group members, while class 1 has a majority of control group instances. These two mining sessions provided additional information about the possibility of external variables (in this case, academic history) that contributed to the differences between the control and treatment groups. Additional statistical testing showed that, between the treatment and control groups, the difference in college grade history could not be discounted. A Fisher's Exact Test gives a $p=0.0724$, which was on the verge of a significant difference. Between the data mining session results and this statistical test, it seemed important to check if there was a relationship between the college grade reported and the difference found between the pretest and posttest scores. Paired t-test calculations gave p-values that are inconclusive and not near the significance threshold. Therefore, there was not a significant impact recorded for this external variable. However, it was likely that some relationship between past college performance and performance in the CS1 class existed in some fashion for testing results. This data could not conclusively reveal what that relationship might have been, nor could it exclude the possibility that the treatment curriculum also had an impact on scores.

5.5.1.3 Unsupervised Cluster without Individual Question Scores

	Class 1	Class 2	Class 3	Class 4	Class 5
Res. Score:	0.436	0.38	0.419	0.457	0.38
No. of Inst.	22	1	5	10	1
Cluster Quality:	0.19	0.04	0.14	0.25	0.04

Table 46: Unsupervised Session 3 Clusters

Since, individual question score attributes could very well be redundant with the total scores, removal of these were expected to provide a clearer view of the real impact score values had on the classification process. A similarity score of 37 was used for this data mining session and all other variables were kept constant with the first session described in 5.5.1.1.

This approach yielded two outlier instances and two larger groups, as well as one smaller sized group. One outlier came from the treatment group and the other from the control group. The treatment outliers distinguished themselves by providing very low self-rating data and the control outlier was isolated due to failure to answer many of the self-rating questions. Class 3 consisted of five instances, all from the control group, who showed a marked decline in testing scores. Again, this type of class appeared in previous data mining sessions with occasional changes in the borderline or special cases. Once again, class 1 consisted of persons with lower test scores while class 4 consisted of those with higher test scores. The means and standard deviations for these classes are shown below. Unlike some of the prior data mining sessions, the standard deviation for the posttest scores were a bit tighter for classes 3 and 4, but class 1 continued to cover a broad range.

	Class 1	Class 2	Class 3	Class 4	Class 5	Domain
Pretot1 (mean)	9.23	4.00	11.00	11.30	9.00	9.85
(sd)	1.82	NA	1.23	2.58	NA	2.33
Pretot2 (mean)	9.82	4.00	9.60	11.80	11.00	10.18
(sd)	2.24	NA	2.07	2.82	NA	2.64
posttot (mean)	7.77	12.00	4.40	10.90	9.00	8.28
(sd)	4.13	NA	1.14	1.79	NA	3.83
diff both (mean)	(1.75)	8.00	(5.90)	(0.65)	(1.00)	(1.73)
(sd)	4.09	NA	2.41	1.80	NA	3.96
	c	t	c	t	c	

Table 47: Session Three Attributes

The generated rule set showed the expected rules for class 3 (which collected instances where performance dropped significantly) and class 4 (which collected instances where there was little change in test scores). The majority of instances for class 4 (eight of ten) came from the treatment group. On the other hand, there was a large subset of rules used to gather instances to class 1. Sixteen control group members and only six treatment group members populated this class. Many of the rules exhibited here matched the rules shown in the first data mining session for class 1; however, additional rules for self-rating scores were added.

Class 1	Class 3	Class 4
exmath = 4	-9<=diff<=-6.5	-2<=diff<=2
Exprobsolve=4		
Exprog = 4		
9<=pretot1<=11		
10<=pretot2<=13		
1<=posttot<=8		
-6.5<=diff<=-2		
Colgrade = b		
Premath=4		
And other combinations		

Table 48: Session Three Rules

The addition of self-evaluation data to the already existing rule set indicated that these factors were of secondary importance for classification. The limited change in classes also indicated that there was very little to be learned from individual question scores on the pretest and posttest. Furthermore, most demographic attributes were conspicuous by their absence, with the exception of the previously discussed grade history. Thus, the focus of continued data mining sessions needed to be on scores and grade history.

5.5.1.4 Unsupervised Cluster without Self-Evaluation Attributes

	Class 1	Class 2	Class 3	Class 4	Domain
Res. Score:	0.364	0.405	0.432	0.519	0.34
No. of Inst.	16	17	4	2	39
Cluster Quality:	0.08	0.21	0.29	0.54	

Table 49: Unsupervised Session 4 Clusters

This data mining session isolated the relationship self-evaluation ratings might have had on the cluster results. A similarity score of 37 was used and all other variables were similar to the other data mining sessions. As with the other sessions, there were two large groups and several smaller groups. Once again, class 3 consisted of persons from the control group who showed a large decline between pretest and posttest scores. Class 4 consisted of two treatment instances where grading background and individual questions scores were quite similar. Class 1 consisted of twelve control members and four treatment members, whereas class 2 consisted of nine treatment and eight control instances. Thus, the split was not as pronounced as it had been with the self-evaluation data.

	Class 1	Class 2	Class 3	Class 4	Domain
pretot1 (mean)	8.94	10.88	11.25	5.50	9.85
(sd)	1.77	2.15	1.26	2.12	2.33
pretot2 (mean)	9.25	11.77	9.50	5.50	10.18
(sd)	2.11	2.14	2.38	2.12	2.64
posttot (mean)	5.94	11.06	4.75	10.50	8.28
(sd)	2.65	3.17	1.50	2.12	3.83
diff both (mean)	(3.16)	(0.27)	(5.63)	5.00	(1.73)
(sd)	2.29	3.99	2.87	4.24	3.96
	c	t	c	t	

Table 50: Session Four Attributes

There appeared to be some classification based on test scores and the differences between the pretest and posttest. However, the two larger groups continued to exhibit rather large standard deviations on the posttest and difference in particular. The rules set (which can be seen in the table below) gave more strength to the combined impact of grade history with overall test scores compared to individual test scores or demographic data. The rule set change did indicate that there was some influence exerted by the self-evaluation data in classification and that they could not be completely discounted from the overall equation of factors that influenced the success of persons in the CS1 course.

Class 1	Class 2	Class 3	Class 4
Post4 = 1	Preq2met = psu	Colgrade = c	2 <= diff <= 2
1 <= posttot <= 5	Post1 = 5		4<=pretot2<=4 and 4<=pretot1<=4
Colgrade = b	12<=pretot2<=16 9<=posttot<=16 -1.5<=diff<=7 Colgrade = a CS0 = a CS0 = a and 9<=posttot<=16		

Table 51: Session Four Rules

5.5.2 Supervised Learning Analysis

Supervised learning looks for patterns that give particular classifications for a given attribute, while unsupervised clustering has no such restriction placed upon it. Supervised learning analysis on the data collected for this study focused on the attempt to generate rules to classify between the control and the treatment group. In each session, two-thirds of the data was used to train the data-mining tool and to allow it to generate a rule base for placing instances into either the treatment or the control groups. The remaining one-third of the instances was used to test the resulting rule base. Data were sorted by identification number so that selection of training instances approached random selection. Also, the first two-thirds of the members from each group were used to train. For example, ten of the fifteen treatment group instances were used in training and five were used to test the results. Three data mining sessions were selected as a representative sample of results for this report.

5.5.2.1 Pretest & Posttest Totals as Learning Input

	Class c1	Class t	Domain
Res. Score:	0.596	0.444	0.54
No. of Inst.	16	10	26
Class Significance:	0.11	(0.17)	

Table 52: Supervised Session One

This data mining session focused only on pretest and posttest total scores and the difference between these scores. This limited data set was used to determine if the test scores alone could classify between the control and treatment groups. Class resemblance scores (shown above) indicated that members of the treatment group training set were actually less similar than the group as a whole (0.444 resemblance score versus 0.54 for the entire group). This was viewed as an indication that the attempt to classify based on this subset of attributes would likely not be successful.

Rules for both classes were generated using the iData Analyzer, first by allowing it to use all instances in the test set to generate the rules, and then by using only the most typical instances to generate the rule set. When all instances were used, the only rule produced was for the control group. This rule prescribed a range for the pretest correctness score ($10 \leq \text{pretot1} \leq 13$) for the control group with a rule accuracy of 76.92%. Using only the most typical instances, a rule was generated for each class. The control group rule was similar to that created before, but with a tighter pretest correctness bound ($10 \leq \text{pretot1} \leq 11$). The treatment group generated a rule based on the difference between pretest and posttest scores ($-2.5 \leq \text{diff} \leq -1.5$) that illustrated minimal change in score between the pretest and posttest. This was the only rule that had any link to the posttest score for this data mining session.

<u>Confusion Matrix</u>		
	Computed Class	
	c1	t
c1	7	1
t	3	2
Percent Correct:		69.0%

Table 53: Confusion Matrix - Supervised One

The error rate was 31% of the test instances fed into the generated model. In particular, this model had difficulty with treatment cases, misplacing three of five in the control group. In fact, this model tended to favor placement in the control group. This made some sense, since the generated rule favored persons with average scores in pretest correctness, so most persons fell into this range. The calculated upper error bound was 56.7%, which meant the accuracy of this model was no better than a toss of the coin for classification of future instances. Much of this was a product of the relatively small number of instances available and the wide range of possible factors influencing the data.

5.5.2.2 Totals and Individual Test Scores as Input

	<u>Class c1</u>	<u>Class t</u>	<u>Domain</u>
Res. Score:	0.285	0.287	0.28
No. of Inst.	16	10	26
Class Significance:	0.03	0.04	

Table 54: Supervised Session Two

Total scores appeared to be inadequate predictors; therefore individual test scores were added to the input data set to determine if there were results within the test scores that had more influence than the total scores. With these additional attributes, the class resemblance scores were much lower, indicating a broader diversity across the wider range of attributes. However, both the control and treatment classes maintained resemblance scores that were similar to the domain resemblance. This made it more likely that there could, in fact, be some differentiation between the classes.

Control Group	Treatment Group
Pretest Quest 2 Clarity = 3	-2.5 <= Difference <= 2.0
Posttest Quest 4 = 1	
10 <= pretot1 <= 13	
10 <= pretot2 <= 13	
3 <= posttot <= 10	
-9 <= difference <= -2	

Table 55: Supervised Two Rule Set

Six rules were generated for the control group and one was generated for the treatment group. The pretest scores in the midrange (for the control group) appeared in the third and fourth rules generated. However, the most interesting information was that those with a larger drop between pretest and posttest scores (rule 6 for control group) were placed in the control group. Those with scores that tended to show little change (rule 1 for the treatment group) were placed in the treatment group. Also, those who scored poorly on the posttest (scores between three and ten inclusive) tended to belong to the control group with this rule set.

Rule 2 provided some interesting insight into the data by claiming that a person scoring one of five points on the fourth question in the posttest would likely be a control group member. The rule accuracy for this rule was higher than all those below it (81.82% rule accuracy), which indicated that persons who did very poorly on this problem tended to be control group members.

Confusion Matrix		
	Computed Class	
	c1	t
c1	6	2
t	1	4
Percent Correct:		76.0%

Table 56: Confusion Matrix - Supervised Two

Application of this rule set to the test group provided a much more successful placement of instances into the proper classes. One of the control group instances was shifted from control to treatment, but two of the incorrectly placed treatment instances were correctly placed in this session. The upper bound error was 47.7%, which was substantially better than the first data mining session. However, this bound also indicated that these results were far from conclusive.

5.5.2.3 Totals, Demographics and Self-Evaluation Data as Input

	Class c1	Class t	Domain
Res. Score:	0.36	0.388	0.36
No. of Inst.	16	10	26
Class Significance:	0.01	0.08	

Table 57: Supervised Session Three

This data mining session used all fields as input except for the individual question scores in the pretest and posttest. And, as with the other sessions, the resemblance scores for the two classes were not significantly different from the overall domain resemblance score. However, it was extremely interesting to note that the rule set had strong similarities to the rules generated in the session reported in section 5.5.2.2. All of the test score related rules appeared as they did in that session with similar accuracy ratings. An additional rule was added to the treatment group rule set for reported 'A' average grades in college. For the control group, two rules indicated that members tended to report high comfort (a four rating out of a one to five scale) with problem-solving and programming. This was interesting considering this same group also generated a rule indicating a tendency to score poorly on the posttest (rule 5 for the control group). The final rule for the control group was that the gender of members tends to be male. This rule was relatively weak, given an accuracy of 66.7%, which was a reflection of the overall demographics for participation.

Control Group	Treatment Group
Exit Prob Solve = 4	-2.5 <= Difference <= 2.0
Exit Programming = 4	College grade = 'a'
10 <= pretot1 <= 13	
10 <= pretot2 <= 13	
3 <= posttot <= 10	
-9 <= difference <= -2	
Gender = M	

Table 58: Supervised Three Rule Set

The confusion matrix for this rule set showed that it was the least successful in categorizing the remaining instances of the three reported data mining sessions.

Confusion Matrix		
	Computed Class	
	c1	t
c1	3	5
t	1	4
Percent Correct:		53.0%

Table 59: Confusion Matrix - Supervised Three

This model tended to favor placing persons into the treatment group rather than the control group. The rule set was similar in rules created for test scores in the previous data mining session, so it was sensible to conclude that the new rules were the primary source for confusion in classification. The upper error bound for this supervised learning session was 74.7%, which indicated that this approach was potentially worse than a simple random selection.

5.6 Qualitative Analysis

5.6.1 Pretest and Posttest Qualitative Data

The qualitative data reported in Chapter 4 provided some interesting information about the tendencies of participants to handle the testing situations. For example, the pretest was administered before any programming had been covered in the CS1 course. Therefore, students were forced to choose their own method of approaching the problems in the pretest and exhibiting their solution and solution process. Qualitative data was collected to determine if there was a difference in approach between the control and treatment groups. The posttest, on the other hand, required that the student provide a programming solution, so there was no longer the broad variation in choice of presentation. However, data were collected regarding any and all additional information provided by the participant on the test as they worked through and presented their solutions.

The treatment group exhibited a trend towards using pseudocode with consistency on the first two problems in the pretest. Although the control group did exhibit a tendency to use pseudocode on the first problem, the quality of this pseudocode tended to be poorer. In fact, many in the control group simply attempted to provide a solution by writing a paragraph describing their ideas. In contrast, only one person used this approach in the treatment group for the first two problems. All but one person used pseudocode as at least part of the description in the treatment group for question one, compared to eight for the control group. Therefore, it was reasonable to conclude that the treatment group felt more comfortable with using this tool to describe algorithmic solutions.

The second significant trend in the pretest was the absence of indecipherable scribbles provided as evidence of work for question four for

the treatment group. Many in the control group (six persons, or 33% answering the question) provided scrawls or treatments that were impossible to get much meaning from as evidence of their work on the problem. On the other hand, the treatment group tended to provide pictures and words that could be more easily deciphered. There were no instances where a question in the treatment group was marked as having indecipherable supporting materials on the pretest.

In the posttest, there were also two significant trends. The first was the use of non-coded solutions in lieu of coded solutions on the posttest by members of the control group. There were three individuals who accounted for all five instances of this occurrence, and in each case, it seemed obvious that the individual was not able to provide even a start to a programming solution to the problem. Thus, these individuals opted to show problem-solving knowledge without exhibiting programming knowledge in these cases. On the other hand, no such instances of substituting programming solutions with words or other options occurred in the treatment group. All treatment group solutions included some component of programming in the target language.

The second posttest trend was the tendency of treatment group members to use pictures to supplement their programming efforts. The control group exhibited one instance of a picture as supplemental information, one with pseudocode, and two with additional words. The treatment group exhibited seven instances of additional pictures and three instances of additional words to supplement the programming solution. In these cases, programming solutions accompanied the additional materials. In fact, it appeared evident that most additional information was intended to aid the student in determining the programming solution, rather than to impress the individual marking the posttest. This appeared even more evident

given the fact that these tests were not graded events and thus had no impact on their course grade.

5.6.2 Demographic Tool and Exit Survey Qualitative Data

Questions concerning the best and worst parts of CS0 uncovered opinion patterns that were encouraging for the improved coverage of problem-solving and algorithm coverage in that class. These comments became stronger at the end of the CS1 course, which increased their validity (as they were voiced after participants had experienced the CS1 course). Regardless of the group, members tended to single out algorithm development and problem-solving as key components of the CS0 experience. However, every member of the treatment group made such a comment in the exit survey, as opposed to seventeen of twenty-four members of the control group. Thus, it is reasonable to state that the treatment group was wholly convinced that these topics were applicable and useful to their programming tasks.

On the negative side, a subset of individuals in the control group isolated the section of curriculum where brief overviews of programming languages were given. None of the treatment group mentioned this section, which may indicate that the instructor used that segment to occupy time that was filled in the treatment group with the new curriculum. Also, one participant noted that the group work section provided in the new curriculum was not (in their eyes) complete. In retrospect, the researcher would tend to agree with this opinion, but it did sufficiently round out the curricular segment for the purposes of this study.

Perhaps the most significant result was that there was no mention of algorithms, pseudocode, diagramming, or problem-solving in comments outlining the least useful parts of CS0. Obviously, since most individuals were citing all or part of this subset as a positive, it would have been odd

to see negative comments here. Individuals in each class made it clear that they felt the entire CS0 class was a waste of their time; however, even these individuals chose not to select these topics as negatives, and many chose to list them instead as positives.

When asked to provide ideas for improving the curriculum, some pointed comments by the control group indicated that they would have liked more material on problem-solving. There was one notable exception that stated a strong belief that CS0 should be skipped and that students should go right to programming. This statement was counterbalanced by the number of persons who stated that their fear for CS1 was that they would not be able to get a handle on all of the syntax of the C++ programming language. These individuals would benefit most by separating the learning curve for beginning problem-solving skills from that of learning programming syntax.

On the other hand, there were few comments suggesting improvement made by the treatment group, and none of them asked for more problem-solving. In fact, their comments tended to be much more specific to events in the CS0 class and less general in nature than those provided by the control group. This might indicate that students felt the class had a more cohesive purpose and order and that they were aware of why topics were covered in the fashion they were presented.

The final point to be made is that there did not appear to be any strong undercurrent with respect to the teacher, the learning environment, and/or events beyond the control of this study. There was one individual who sang the praises of their instructor in the CS0 course, but since this instructor was the same for both groups, it would not have impacted results even if several students had felt that way. There were some hints that perhaps there were topics covered in the control group CS0

experience that were not in the treatment group CS0 experience (such as introductions to various languages). However, since the researcher was assured that there were no substantive changes made other than the curriculum submitted to BSU, it seemed reasonable to assume that any unreported changes were minor adjustments, as the instructor did not see fit to mention them.

5.7 Interpretation of Findings

The most pertinent and important findings of this study are summarized below. Details about each may be found in the prior analysis sections in this chapter and in the data reporting section in Chapter Four.

- Posttest scores between the control and treatment groups did not indicate a significant difference between the two ($p = 0.089$). But there was an observed difference, with the treatment group scoring higher on the posttest.
- Pretest correctness scores between the control and treatment groups did not show statistically significant similarity ($p = 0.924$)
- Treatment group programming self-evaluation scores were significantly different at the end of the CS1 from the scores received at the beginning of the CS1 class ($p = 0.023$). Similar scores for the control group were not significantly different.
- The self-evaluation scores for programming at the end of the CS1 course were statistically similar for the treatment group and the control group ($p = 0.999$).
- The satisfaction scores for the CS0 course were significantly different between the two groups with p values below the threshold.
- Unsupervised clustering data mining sessions developed rules and classifications largely based on test scores and self-reported college grades.

- Supervised learning data mining sessions confirmed rules based on test scores, but showed increased confusion when college grades were added to the rule base.
- Data mining results based on test score rules illustrated a tendency for lower posttest scores and larger drops in scores between the pretest and posttest for the control group.
- Pretest qualitative information illustrated a greater tendency by the treatment group to use pseudocode.
- Pretest qualitative information showed a tendency for control group members to use indecipherable scribbles, whereas the treatment group did not exhibit this.
- Posttest qualitative information uncovered a tendency for members of the treatment group to use pictures to supplement coding development.
- Posttest qualitative results included instances where control group members provided non-programming answers on a programming test, but there were no such instances for the treatment group.
- Open-ended questions used to isolate positive and negative aspects of the CS0 course uncovered very strong support for problem-solving, algorithms, pseudocode and diagramming in both groups which increased on completion of the CS1 course.
- Open-ended questions emphasized student concern that they would not be able to master all of the pertinent programming language syntax necessary in the CS1 course.

These findings are interesting in that they tend to support each other in several ways. First, it seems clear that members of the two groups tended to approach the tests in different ways. Second, participants between the two groups appeared to have a different feeling for their CS0 class experience. Third, members of both groups gave themselves identical comfort level ratings for programming (despite different starting points) at

the end of the CS1 course. And finally, members of the treatment group tended to score better on the programming posttest than members of the control group (although this difference is just outside the threshold for statistical significance).

These results support Subordinate Hypothesis number one, which claimed that these changes to the curriculum would result in a measurable increase in satisfaction for the applicability of the CS0 course. Members of the treatment group gave ratings that were, on average, a full point higher on a five-point scale for this course. This was directly supported by the generally positive comments for the course from the treatment group, as compared to the highly qualified and particular negative comments. On the other hand, the control group was much more liberal with criticisms for the course. Thus, it can be stated that the curricular changes had some impact on improving satisfaction in the experience.

These results do not support Subordinate Hypothesis number two as it was intended to be measured. This hypothesis claimed that there would be an increase in the self-evaluation rating for programming with the new curriculum. Although there was a significant change in confidence for the treatment group between the beginning and end of the CS1 course, an increase in confidence between groups cannot be demonstrated. In fact, the scores were statistically similar at the end of the course for both groups. However, the treatment group began with less programming experience than the control group, allowing them to exhibit a significant change over time. Overall, however, this subordinate hypothesis must be rejected.

The overall null hypothesis, that there would be no measurable difference in the exhibited learning of programming skills between the two groups, was also rejected. In particular, it should be noted that pretest correctness

scores were very nearly statistically identical, whereas posttest scores were very nearly statistically different between groups. This indicates that both groups had fairly similar natural skills at the beginning of the course. However, the second group fared better in applying those skills to programming. Data mining results also supported the tendency to differentiate between groups based on posttest scores, particularly on the drop exhibited between pretest and posttest scores by the control group. Finally, qualitative data illustrated that members of the treatment group actually used tools provided them in CS0 in order to support their work in both the pretest and posttest. This alone exhibited a difference in how they approached the problem of learning to program.

This study was, however, unable to extend the results to the alternate hypothesis, which stated that there would be a measurable change in programming ability between the groups. While there were observable differences, they were not statistically significant within an acceptable threshold of probability. Furthermore, there were simply too many outside variables that could not be fully accounted for. For example, although data mining results indicated limitations on the impact of differences in historical college grades reported, the possibility that prior successes led to additional success in learning programming cannot be dismissed.

In addition, there were numerous other factors that could not be accounted for in this study, such as teaching differences from one semester to another. For example, it is entirely possible that results in this study were partly a product of the 'Hawthorne effect,' observed in instances where an instructor's own enthusiasm for a new approach or material is conveyed to students. This kind of impact was lessened somewhat by the researcher's distance from the practicing instructor. Furthermore, participating instructors were not recruited for their willingness to agree or disagree with the researcher on pedagogy. It is

even possible that the instructor's reduced familiarity with the new material might have led to some confusion and discomfort in the classroom, which would have mitigated against the Hawthorne effect.

Other external variables were collected as demographic data in this study and subjected to statistical tests and data mining sessions to determine if these factors might have influenced the results. The only external variable that appeared to have any influence over the results was the self-reported average college grade, or the self-reported CS0 grade. Other variables were successfully eliminated with statistical tests. This was confirmed by the corresponding lack of such information in the rule sets for the data mining sessions.

5.8 Summary of Findings

5.8.1 Extensibility of the Study

It has been shown that, given the sample population at Bemidji State University and the sample population of undergraduate computer science programs, the sample for this study was representative of a typical small to mid-range post-secondary school in the Midwestern United States. Both the control and treatment groups came from all possible candidates for the computer science program at BSU during the academic terms of the study, so no self-selection was possible. On the other hand, the sample size was small enough that the application of the results cannot be extended beyond sample populations that resemble the population at Bemidji State University. Therefore, this researcher concludes that altering current computer science programs in similar environments to include increased and focused coverage of problem-solving, algorithm development, pseudocode, and diagramming should result in improved programming learning for participants.

These results may be extensible to a broader population, but research with a broader sample would have to be undertaken to investigate this possibility. It is also acknowledged that similar institutions might find divergent results depending on the surrounding curriculum, willingness of faculty to implement changes, and environmental variables. The strongest statement that can be made as a result of this study is that computing programs should consider covering problem-solving, algorithm development, pseudocode, and diagramming prior to programming. The results of this study indicate that the outcome would likely be (at worst) similar to programming first approaches and (at best) better than a programming first approach.

The data suggest that the effectiveness of these curricular changes vary depending on individual student characteristics. This is not surprising, of course, since learning is a personal endeavor and it would be inappropriate to suggest that one approach is sufficient for all persons. This study did not attempt to break the population into subgroups, but there is sufficient information to suggest that certain types of students may benefit more from this approach than others.

5.8.2 Results in Context of Body of Knowledge

The results of this study confirm or extend what is currently understood within the computer science education body of knowledge for introductory courses. Self-efficacy studies such as those by Quade (2003) and Ramalingam and Wiedenbeck (1998) linked program-solving to computer science and to programming. This study suggests a link between teaching problem skills specific to computer science and programming and success in improving early programming skills. Other studies by Applin (2001) and Bailie (1991) argued that there are learning advantages when advanced organizers are provided to learners. In fact, Proulx (2000) argued that problem-solving concepts are lost and syntax takes over when

the programming language is the initial problem-solving tool for many students. The curricular changes outlined in this study provided advanced organizers for programming that appeared to aid in learning. Also, tools were provided that allowed the focus to remain on problem-solving concepts rather than syntax during early learning.

Control group members appeared to have fewer opinions on the applicability of the CS0 course and were less satisfied with it; the treatment group was comfortable with the goals and applicability of the course. This correlates with works such as that provided by Powers and Powers (2001), which claim that inaccurate notions about purpose and goals in learning can be detrimental to future learning. A stronger sense of what computer science is all about and the place their learning has within the whole encourages students to continue in the program (Sanders and Mueller, 2000). Students who have a better sense of their learning goals tend to do a better job following through with their learning (Black and Deci, 2000). Furthermore, treatment group participants illustrated their ability to draw connections between CS0 subject material and CS1 subject material by exhibiting more instances of applying diagramming or pseudocode solutions on the pretest and posttest. Pedagogically, this supported both spacing and spiraling, which support learning (Powers, 2002).

This research does not provide direct support for or against breadth-first, breadth-also or depth-first approaches to CS0, although the generally positive results do indicate that the breadth-first model can become more successful with careful consideration of the courses surrounding it in the program. Furthermore, studies such as that provided by Stein (2002) show a link between success in an introductory course leading to success in subsequent courses in a sequence, so increased success in CS0 should certainly increase the likelihood of success in CS1. In fact, this

study provided some evidence that persons doing well in CS1 had received outstanding grades in CS0. The positive results of this research also indicate that using a spiraling approach (where problem-solving concepts are introduced outside of programming and then reintroduced in programming) could be effective, regardless of the curricular model.

It could be argued that the modified curriculum prepared students for future success in computer science learning by allowing them to construct a framework for that success. The pretest scores for both groups were statistically similar, indicating that raw problem-solving abilities were roughly equivalent. However, CS0 reported grades were different. An argument could be made that students were given a better framework from which to work that encouraged their learning in CS0 (Cook, 1997). With success in the first course, and a strong link to the second course, students were able to set an expectation for success in CS1 (Bay and Daniel, 2003).

Finally, regardless of how well they fit cited studies and accepted pedagogical practices, the results of this study must be qualified based on the size of the sample and limitations on extensibility. Furthermore, the number of external variables that might have had an impact on the study results, despite attempts to isolate and account for them, makes it difficult to state claims with certainty. However, it is evident that there was some difference between the control and treatment groups and it is evident that improvement was shown by the treatment group. When the consistency of results between statistical, qualitative, and data mining analysis is considered, it becomes difficult to ignore the fact that something positive occurred in the course where the new curriculum was applied.

5.9 Recommendations for Future Research

As is true for most field research and education research, this study leaves the researcher with more questions than answers. The most important issue is exactly what it was about the curriculum that led to an improvement in learning. Was it a renewed focus on the entire CS0 course, or was it increased coverage of problem-solving and algorithm development? Would similar modifications of content in a depth-first environment lead to improvements in programming learning? In other words, is this link independent of curriculum strategy? And, perhaps the most interesting question, were certain kinds of students helped more by these changes than others? And, if so, who were they and how can others like them be reached?

If similar research were to be undertaken at this point in time, it would be useful to utilize self-efficacy scales developed for programming and computer science in order to have a tool that has been validity and reliability tested for determining predispositions in learners. This would allow more accurate accounting for variables that influence learning not related to the change being measured. Similarly, known measures of critical thinking or problem-solving ability, such as the Watson-Glaser tool (Watson, Glaser, 1994), could be used to set baseline values for ability. Also, the emphasis could be moved from timed measurements and tests of skills to qualitative and quantitative studies of actual programs written by subjects of the study. The evaluation methods outlined by Mengel (1999) or Howatt (1994) might be useful in such an endeavor.

Future research in this area could also focus on the extensibility of the research. Various approaches to expand the study to multiple schools with a diverse set of populations could be helpful (Sandstrom and Daniels, 2000). A study following a cohort through their college career could

assess how different introductory methods impact students as they travel through their computing career at given four-year degree granting institutions. Any future studies of this sort would require more direct supervision and increased access to resources. There would also have to be programs willing to execute changes to their introductory curriculum in a controlled fashion in order for changes to be adequately isolated for measurement.

It is obvious that this field of research has many open questions. It is equally clear that post-secondary programs are still seeking out best practices for the introductory sequence in computer science. A long-term research agenda in this area (that paid increased attention to isolating variables and confirming study validity and reliability) would be beneficial to students and educators alike.

6 Bibliography

- ACM/IEEE-CS Joint Curriculum Task Force. "Computing Curricula 1991." : Association For Computing Machinery, 1991.
- ACM/IEEE-CS Joint Curriculum Task Force. "Computing Curricula 2001." : Association for Computing Machinery, 2001.
- Ackoff, R. *The Art of Problem-solving*. New York: John Wiley & Sons, 1987.
- Adams, J. *Conceptual Blockbusting*. San Francisco, CA: W.H.Freeman, 1974.
- Aharoni, D. "Cogito, Ergo Sum!: Cognitive Processes of Students Dealing with Data Structures." Paper presented at the SIGCSE, Austin, TX, March 2000.
- Akingbade, A, T Finley, D Jackson, P Patel, and S Rodger. "JAWAA: Easy Web-Based Animation from CS0 to Advanced CS Courses." Paper presented at the SIGCSE 2003, Reno, NV, Feb 2003.
- Albers, D.J., Anderson, R.D. & Loftsgaard, D.O. *Undergraduate Programs in the Mathematical and Computer Science: the 1985-86 Survey*. Washington, DC: Math Association of America, 1987.
- Almstrum, V. "Investigating Student Difficulties with Mathematical Logic." In *Teaching and Learning Formal Methods*, edited by N Dean and M Hinchey: Academic Press, 1996.
- Almstrum, V, O Hazzan, D Ginat, and J Clement. "Transfer to/from Computing Science Education: The Case of Science Education Research." Paper presented at the 8th Annual Conference on Innovations and Technology in Computer Science Education, Reno, NV, Feb 19-23, 2003.
- Almstrum, V, O Hazzan, D Ginat, and T Morley. "Import and Export to/from Computing Science Education: The Case of Mathematics Education Research." Paper presented at the 7th Annual Conference of Innovations and Technology in Computer Science Education, Aarhus, Denmark, June 24-28, 2002.
- Apple, D, and D Nelson. "Identification of Non-Success Factors in a Large Introductory Computer Science Course and Constructive Interventions for Increasing Student Success." Paper presented at the 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, Nov 6-9, 2002.
- Applin, A. "Second Language Acquisition and CS1: Is * = = ** ?" Paper presented at the SIGCSE 2001, Charlotte, NC, Feb 2001.
- August, R, G Lopez, C Yokomoto, and W Buchanan. "Heuristic Beliefs About Problem-solving in Technology Courses and Their Impact on Success on Problem-solving Exams." Paper presented at the 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, Nov 6-9, 2002.
- Ausubel, D. *Educational Psychology: A Cognitive View*. New York: Holt, Rinehart and Winston, 1968.

- Bagert, D, W Marcy, and B Calloni. "A Successful Five-year Experiment with a Breadth-first Introductory Course." *SIGCSE Bulletin* 27, no. 1 (1995): 116-120.
- Bailie, F. "Improving the Modularization Ability of Novice Programmers." Paper presented at the 22nd SIGCSE Technical Symposium on Computer Science Education 1991.
- Bandura, A. *Social Foundation of Thought and Action*. Englewood Cliffs, NJ: Prentice Hall, 1986.
- Barr, M, S Holden, D Phillips, and T Greening. "An Exploration of Novice Programming Errors in an Object-Oriented Environment." *SIGCSE Bulletin* 31, no. 4 (1999): 42-46.
- Bay, D, and H Daniel. "The Theory of Trying and Goal-Directed Behavior: The Effect of Moving Up the Hierarchy of Goals." *Psychology and Marketing* 20, no. 8 (2003): 669-684.
- Ben-Ari, M. "Constructivism in Computer Science Education." Paper presented at the SIGCSE 1998, Mar 1998.
- Ben-Ari, M. "Constructivism in Computer Science Education." *Journal of Computers in Mathematics and Science Teaching* 20, no. 1 (2001): 45-73.
- Berlyne. "Curiosity and Education." In *Learning and the Education Process*, edited by J Krumboltz, 67-89. Chicago: Rand McNally, 1965.
- Black, A, and E Deci. "The Effects of Instructors' Autonomy Support and Students' Autonomous Motivation on Learning Organic Chemistry: A Self-Determination Theory Perspective." *Science Education*, no. 84 (2000): 740-756.
- Bloom, B.S., B.B. Mesia, and D.R. Krathwohl. *Taxonomy of Educational Objectives*. 2 vols. Vol. 1 & 2. New York: David McKay, 1964.
- Booth, S. "On Phenomenography, Learning and Teaching." *Higher Education Research and Development* 16, no. 2 (1997): 135-158.
- Boud, and Miller. "Synthesizing Traditions and Identifying Themes in Learning from Experience." In *Working with Experience: Animating Learning*, edited by Boud and Miller, 14-24. London: Routledge, 1996.
- Bouvier, D. "Pilot Study: Living Flowcharts in an Introduction to Programming Course." Paper presented at the SIGCSE 2003, Reno, NV, Feb 19-23, 2003.
- Brookfield, Stephen D. *The Skillful Teacher: On Technique, Trust, and Responsiveness in the Classroom*. 1st ed. San Francisco, CA: Jossey-Bass Inc., Publishers, 1991.
- Brookshear, J.G. *Computer Science: An Overview*. 6th ed. 1 vols. Reading, MA: Addison-Wesley, 2000.
- Brown, J, and G Dobbie. "Supporting and Evaluating Team Dynamics in Group Projects." Paper presented at the 24th SIGCSE Technical Symposium on Computer Science Education, Mar 1999.
- Bruffee, Kenneth A. *Collaborative Learning: Higher Education, Interdependence, and the Authority of Knowledge*. Baltimore, MD: The Johns Hopkins University Press, 1993.

- Bryant, R, and M Vardi. "2000-2001 Taulbee Survey." *Computing Research News*, Mar 2002, 4-11.
- BSU. *University Data Book* [website]. Bemidji State University, 2003 [cited Oct 5, 2003]. Available from <http://www.bemidjistate.edu/InstRes/databook/tableofcontent.html>.
- Buck, D, and D Stucki. "Design Early Considered Harmful: Graduated Exposure to Complexity and Structure Based on Levels of Cognitive Development." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar, 2000.
- Burt, C. "The Structure of the Mind." In *Intelligence and Ability*, edited by S. Wiseman, 193-217. Baltimore, MD: Penguin Books, 1967.
- Butcher, D, and W Muth. "Predicting Performance in an Introductory Computer Science Course." *Communications of the ACM*, March, 1985, 263-268.
- Byrne, P, and G Lyons. "The Effect of Student Attributes on Success in Programming." Paper presented at the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Caterbury, UK, Jun 2001.
- Carbone, A., J. Hurst, I Mitchell, and D Gunstone. "Characteristics of Programming Exercises that lead to Poor Learning Tendencies: Part II." Paper presented at the 6th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Canterbury, UK, June 2001.
- Carrano, F.M. *Data Abstraction and Problem-solving with C++: Walls and Mirrors*. 1 vols. Redwood City, CA: Benjamin/Cummings Publishing Company, Inc., 1995.
- Carrasquel, J. "Teaching CS1 On-line: the Good, the Bad and the Ugly." Paper presented at the SIGCSE Technology Symposium on Computer Science Education, 1999.
- Carter, J. "Collaboration or Plagiarism: What Happens When Students Work Together." Paper presented at the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Cracow, Poland, Jun 1999.
- Center, State Demographic. *Census 2000 Data* [web]. State Demographic Center, 2003 [cited Oct 5, 2003]. Available from <http://www.demography.state.mn.us/>.
- Chamillard, A. "Using Learning Style Data in an Introductory Computer Science Course." Paper presented at the 13th SIGCSE Technical Symposium on Computer Science Education 1999.
- Chamillard, A, and K Braun. "Evaluating Programming Ability in an Introductory Computer Science Course." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar 2000.
- Chang, K, B Chiao, S Chen, and R Hsiao. "A Programming Learning System for Beginners - A Completion Strategy Approach." *IEEE Transactions on Education* 43, no. 2 (2000): 211-220.

- Christophel, D.M., and J. Gorham. "A Test-retest Analysis of Student Motivation, Teacher Immediacy, and Perceived Sources of Motivation and Demotivation in College Classes." *Communication Education* 44 (1995): 292-306.
- Clancy, M, J Stasko, M Guzdial, S Fincher, and N Dale. "Models and Areas for Computer Science Education Research." *Computer Science Education* 11, no. 4 (2001): 323-340.
- Comer, J, and R Roggio. "Teaching a Java-based CS1 Course in an Academically-Diverse Environment." Paper presented at the SIGCSE 2002, Covington, KY, Mar 2002.
- Concepcion, A.I., L.E. Cummins, E.J. Moran, and M.M. Do. "Algorithma 98: An Algorithm Animation Project." Paper presented at the SIGCSE, New Orleans, LA, March 1999.
- Cook, C. "CS0: Computer Science Orientation Course." Paper presented at the 28th SIGCSE Technical Symposium on Computer Science Education, Feb 1997.
- Cook, D. "The Impact of the Hawthorne Effect in Experimental Design in Educational Research." In *US Office of Education, NO0726*. Washington, DC: US Office of Education, 1967.
- Cook, T. D., and D. T. Campbell. *Quasi-Experimentation: Design & Analysis Issues for Field Settings*. Boston, MA: Houghton Mifflin Company, 1979.
- Cook, T.D. , and C.S. Reichardt, eds. *Qualitative and Quantitative Methods in Evaluation Research*. 1 vols. Beverly Hills, CA: Sage Publications, 1979.
- Cowling, A.J. "Structuring the Recurring Concepts in the SE Curriculum." *Forum for Advancing Software engineering Education* 9, no. 2 (1999): 6-14.
- Davy, J, and T Jenkins. "Research-led Innovation in Teaching and Learning Programming." Paper presented at the 4th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Cracow, Poland, Jun 1999.
- Decker, R. "The Case for CS0." *Computer Science Syllabus*, Nov 1992 1992, 5-7.
- Decker, R, and S Hirschfield. *The Analytical Engine: An Introduction to Computer Science Using Hypercard*: Wadsworth Publishing, 1990.
- Deimel, L, and D Moffat. "A More Analytical Approach to Teaching the Introductory Programming Course." Paper presented at the NECC 1982.
- Dempster, F. "The Spacing Effect: A Case Study in the Failure to Apply the Results of Psychological Research." *American Psychologist* 1988, 627-634.
- Deutsch, M. "A Theory of Cooperation and Competition." *Human Relations* 2 (1949): 129-152.
- Dewey, John. *Experience and Education*. reprinted ed. New York, NY: MacMillan, 1963.
- Dingle, A, and C Zander. "Assessing the Ripple Effect of CS1 Language Choice." *Journal of Computing Sciences in Colleges* 16, no. 2 (2001): 85-94.

- Diseth, A. "Personality and Approaches to Learning as Predictors of Academic Achievement." *European Journal of Personality*, no. 17 (2002): 143-155.
- Dressel, P.L. *Handbook of Academic Evaluation*. 1 vols. San Francisco, CA: Jossey-Bass, 1976.
- Dunham, M.H. *Data Mining: Introductory and Advanced Topics*. Upper Saddle River, NJ: Prentice Hall, 2003.
- Eble, K.E. *The Craft of Teaching: A Guide to Mastering the Professor's Art*. 2nd ed. San Francisco, CA: Jossey-Bass, 1990.
- Edinger, M. "Sequence and Scope in the Curriculum." *FT Magazine: Education* 117, no. 1 (1996): 58+.
- Ericson, B, and E Rogers. "Interactive Student Support for Introductory Computer Science Courses." Paper presented at the ASEE/IEEE Frontiers in Education Conference 1996.
- Eyck, J.T., G Sampath, and R Goldstone. "Specification of an Algorithm Design System." Paper presented at the ITiCSE, Dublin, Ireland, 1998.
- Fekete, A, J Kay, J Kingston, and K Wimalaratne. "Supporting Reflection in Introductory Computer Science." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar 2000.
- Felleisen, M. "A Design-based Introductory Computer Science Course." Paper presented at the 1997 ASEE/IEEE Frontiers in Education Conference 1997.
- Filipovitch, A. *IRB Brochure* [website]. Minnesota State University at Mankato, [cited August 23, 2002]. Available from <http://www2.mnsu.edu/graduate/facstaff/irbBrochure.shtml>.
- Fischbein, E. *Intuition in Science and Mathematics*. Dordrecht, Holland: D Reidel Publishing Company, 1987.
- Flaherty, T.J. *Institutional Review Forms* [website]. Minnesota State University - Mankato, [cited August 23, 2002]. Available from <http://www2.mnsu.edu/graduate/facstaff/irbForms.shtml>.
- Frazer, K. "Comparing the Impact of Two Assignment-based Teaching Methodologies on Student Programming." *Journal of Computer Science Education* (1998): 21-26.
- Fujii, T. "The Role of Cognitive Conflict in Understanding Mathematics." Paper presented at the PME-11 1987.
- Gibbs, D. "The Effect of a Constructivist Learning Environment for Field-Dependent/Independent Students on Achievement in Introductory Computer Science." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar 2000.
- Ginat, D. "Misleading Intuition in Algorithmic Problem-solving." Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education, Charlotte, NC, Feb 2001.

- Ginat, D, O Astrachan, D Garcia, and J Bergin. "Colorful Illustrations of Algorithmic Design Techniques and Problem-solving." Paper presented at the SIGCSE, Covington, KY, Feb 27 - Mar 3, 2002.
- Glass, R.L. "The Relationship Between Theory and Practice in Software Engineering." *Communications of the ACM* 39, no. 11 (1996): 11-13.
- Goldwasser, M.H. "A Gimmick to Integrate Software Testing Throughout the Curriculum." Paper presented at the SIGCSE, Covington, KY, Feb 27 - Mar 3, 2002.
- Goldweber, M, J Barr, and C Leska. "A New Perspective on Teaching Computer Literacy." Paper presented at the 25th SIGCSE Technical Symposium on Computer Science Education, Mar 1994.
- Goold, A, and R Rimmer. "Factors Affecting Performance in First Year Programming." *ACM SIGCSE Bulletin* 32 (2000): 39-43.
- Gray, C, and M Frazier. "Introducing Computer Science After Programming." *Journal of Computing Science in Colleges* 18, no. 1 (2002): 65-76.
- Gronlund, N.E. & Linn, R.L. *Measurement and Evaluation in Teaching*. 6th ed. 1 vols. New York, NY: MacMillan Publishing, 1990.
- Hagan, D, and S Markham. "Does it Help to Have Some Programming Experience Before Beginning a Computer Degree Program?" Paper presented at the 5th Annual SIGCSE/SIGCUE Conference on Innovation and Technology in Computer Science Education, Helsinki, Finland, Jul 2000.
- Hazzan, O. "Reducing Abstraction Level when Learning Abstract Algebra Concepts." *Educational Studies in Mathematics* 40, no. 1 (1999): 71-90.
- Heron, J. *Group Facilitation: Theories and Models for Practice*. London: Kogan Page, 1989.
- Heron, J. "Helping Whole People Learn." In *Working with Experience: Animating Learning*, edited by Boud and Miller, 75-91. London: Routledge, 1996.
- Heth, C, E Cornell, and T Flood. "Self-ratings of Sense of Direction and Route Reversal Performance." *Applied Cognitive Psychology*, no. 16 (2002): 309-324.
- Hilburn, T. "Software Engineering Education: A Modest Proposal." *IEEE Software* 14, no. 4 (1997): 44-48.
- Hilburn, T. "A Top-Down Approach to Teaching an Introductory Computer Science Course." Paper presented at the 24th SIGCSE Technical Symposium on Computer Science Education 1993.
- Holland, J.L. "Explorations of a Theory of Vocational Choice: VI. A Longitudinal Study Using a Sample of Typical College Students." *Journal of Applied Psychology* 52 (1968): 1-37.
- Holt, R, D Boehm-Davis, and A Schultz. "Mental Representation of Programs for Student and Professional Programmers." In *Empirical Studies of Programmers: Second Workshop*, edited by G Olson, S Sheppard and E Soloway. Norwood, NJ: Ablex, 1987.
- Howatt, J. "On Criteria for Grading Student Programs." *SIGCSE Bulletin* 3 (1994): 3-7.

- Howell, K. "The Culture of Undergraduate Computer Science Education: Its Role in Promoting Equity Within the Discipline." Dissertation, Oregon State University, 1996.
- Hull, C.L. "The Basic Constitution of Aptitude." In *Intelligence and Ability*, edited by S. Wiseman, 90-114. Baltimore, MD: Penguin Books, 1967.
- Hunt, J.M. "Intelligence and Experience." In *Intelligence and Ability*, edited by S. Wiseman, 314-356. Baltimore, MD: Penguin Books, 1967.
- Isaacson, P, and T Scott. "A Comparison Between Python and TCL Solution on Four Programs Assigned in CS0." Paper presented at the Consortium for Computing Sciences in Colleges 2002.
- Jarc, D.J., M.B. Feldman, and R.S. Heller. "Assessing the Benefits of Interactive Prediction Using Web-based Algorithm Animation Courseware." Paper presented at the SIGCSE, Austin, TX, March 2000.
- Johnson, D, R Johnson, and K Smith. *Active Learning: Cooperation in the College Classroom*. Edina, MN: Interaction Book Company, 1998.
- Johnson, D.W., R.T. Johnson, and K.A. Smith. *Cooperative Learning: Increasing College Faculty Instructional Productivity*. Washington, DC: School of Education and Human Development, Washington University, 1991.
- Kagan, S., and M Kagan. "The Structural Approach: Six Keys to Cooperative Learning." In *Cooperative Learning: Theory and Research*, edited by S. Sharan. New York: Praeger, 1990.
- Kelley, D.H., and J Gorham. "Effects of Immediacy on Recall of Information." *Communication Education* 37 (1988): 198-207.
- Knowles, M, and H Knowles. *Introduction to Group Dynamics*. Revised ed. Chicago, IL: Follett Publishing Company, 1972.
- Koffman, E. "CS1 Using Java Language Features Gently." Paper presented at the ITiCSE 1999, Innovation and Technology in Computer Science Education, Jul 1999 Wolz, U.
- Kozen, Dexter, and S Zweben. "1996-1997 CRA Taulbee Survey." *Computing Research News*, Mar 1998, 4-8.
- Krantz, S. *Techniques of Problem-solving*: American Mathematical Society, 1991.
- Larkin, J.H., J.I. Heller, and J.G. Greeno. "Instructional Implications of Research on Problem-solving." In *New Directions for Teaching and Learning: Learning, Cognition and College Teaching*, edited by W.J. McKeachie, 51-65. San Francisco, CA: Jossey-Bass, 1980.
- Leestma, S, and L Nyhoff. *Pascal: Programming and Problem-solving*. New York: Macmillan Publishing Company, 1984.
- Lethbridge, T. "A Survey of the Relevance of Computer Science and Software Engineering Education." Paper presented at the 11th International Conference on Software Engineering 1998.
- Lewandoski, G, and Morehead. "Computer Science Through the Eyes of Dead Monkeys." Paper presented at the SIGCSE 1998, Feb 1998.

- Long, T, B Weide, P Bucci, and M Sitaraman. "Client View First: An Exodus from Implementation-based Teaching." Paper presented at the 13th SIGCSE Technical Symposium on Computer Science Education 1999.
- Marion, W. "CS1: What Should We Be Teaching?" *SIGCSE Bulletin* 31, no. 4 (1999): 35-38.
- Markova, D. *The Open Mind: Exploring the Six Patterns of Natural Intelligence*: Conari, 1996.
- Mayer, R. "The Psychology of How Novices Learn Programming." *Computing Surveys* 1 (1981): 121-141.
- McCauley, R, and U Jackson. "Teaching Software Engineering Early - Experiences and Results." *SIGCSE Bulletin, Inroads: Paving the Way Towards Excellence in Computing Education* 31, no. 2 (1999): 86-91.
- McDowell, C, L Werner, H Bullock, and J Fernald. "The Effects of Pair-Programming on Performance in an Introductory Programming Course." Paper presented at the 33rd SIGCSE Technical Symposium on Computer Science Education, Feb 2002.
- Mengel, S, and V Yerramilli. "A Case Study of the Static Analysis of Novice Student Programs." Paper presented at the 13th SIGCSE Technical Symposium on Computer Science Education, March, 1999.
- Miller, N., and D Boud. "Animating Learning from Experience." In *Working with Experience: Animating Learning*, edited by D. Boud and N. Miller, 3-13. London: Routledge, 1996.
- Mitchell, W. "Another Look at CS0." *Journal of Computer Sciences in Colleges* 17, no. 1 (2001): 194-205.
- Mitchell, W. *Prelude to Programming*. New York: Reston, 1984.
- Morrison, M, and T Newman. "A Study of the Impact of Student Background and Preparedness on Outcomes in CS1." Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education 2001.
- Moss, B. "Creating Integrated Curriculum: Books to Guide the Process." *Reading Teacher* 4, no. 48 (1995): 358+.
- Nagappan, N, L Williams, M Ferzli, E Wiebe, K Yang, C Miller, and S Balik. "Improving the CS1 Experience with Pair Programming." Paper presented at the 34th SIGCSE Technical Symposium on Computer Science Education, Feb 2003.
- Naps, T, G Rossling, V Almstrum, W Dann, R Fleischer, C Hundhausen, A Korhonen, L Malmi, M McNally, S Rodger, and J Valazquez-Iturbide. "Exploring the Role of Visualization and Engagement in Computer Science Education, Report of the Working Group on Improving the Educational Impact of Algorithm Visualization." Paper presented at the 7th Annual Conference on Innovation and Technology in Computer Science Education 2002.
- Newell, A., and H. Simon. *Human Problem-solving*. Englewood Cliffs, NJ: Prentice-Hall, 1972.

- Odekirk, E, D Jones, and P Jensen. "Three Semesters of CS0 using Java: Assignments and Experiences." Paper presented at the ITiCSE 2000, Innovation and Technology in Computer Science Education, Jul 2000.
- Pattis, R. *Karel the Robot: A Gentle Introduction to the Art of Programming*. New York: Wiley and Sons, 1981.
- Perfetti, C. "Levels of Language and Levels of Process." In *Levels of Processing in Human Memory*, edited by L Cermak and F Craik, 159-181. Hillsdale, NJ: Erlbaum, 1979.
- Pike, G. "The Relationship Between Self Reports of College Experiences and Achievement Test Scores." *Research in Higher Education* 36, no. 1 (1995).
- Polya, G. *How to Solve It?* Princeton, NJ: Princeton University Press, 1973.
- Powers, D, and K Powers. "Constructivist Implications of Preconceptions in Computing." Paper presented at the ISECON 2000, Philadelphia, PA, Nov 2000.
- Powers, K. "Breadth-Also: A Rationale and Implementation." Paper presented at the SIGCSE 2003, Reno, NV, Feb 2003.
- Prince, M, and B Hoyt. "Helping Students Make the Transition from Novice to Expert Problem-Solvers." Paper presented at the 32nd ASEE/IEEE Frontiers in Education Conference, Boston, MA, Nov 6-9, 2002.
- Proulx, V. "Programming Patterns and Design Patterns in the Introductory Computer Science Course." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX, Mar 2000.
- Quade, A. "Development and Validation of a Computer Science Self-Efficacy Scale for CS0 Courses and the Group Analysis of CS0 Student Self-Efficacy." Paper presented at the Proceedings of the International Conference on Information Technology: Computers and Communications 2003.
- Ramalingam, V, and S Wiedenbeck. "Development and Validation of Scores on a Computer Programming Self-Efficacy Scale and Group Analysis of Novice Programmer Self-Efficacy." *Journal of Educational Computing Research* 19, no. 4 (1998): 367-381.
- Reed, D. "Rethinking CS0 with Javascript." Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education, Feb, 2001.
- Roberge, J, and C Suriano. "Using Laboratories to Teach Software Engineering Principles in the Introductory Computer Science Curriculum." Paper presented at the 25th Annual SIGCSE Symposium on Computer Science Education 1994.
- Roiger, R.J., and M.W. Geatz. *Data Mining: A Tutorial-Based Primer*. Boston, MA: Addison-Wesley, 2003.
- Rosling, G., and T.L. Naps. "A Testbed for Pedagogical Requirements in Algorithm Visualizations." Paper presented at the ITiCSE, Aarhus, Denmark, June, 24-26,2002.
- Rosso, A, and M Daniele. "Our Method to Teach Algorithmic Development." *SIGCSE Bulletin* 32, no. 2 (2000): 49-52.

- Rountree, N, J Rountree, and A Robins. "Predictors of Success and Failure in a CS1 Course." *SIGCSE Bulletin* 34, no. 4 (2002): 121-124.
- Sandstrom, A, and M Daniels. "Time Studies as a Tool for (Computer Science) Education Research." Paper presented at the the Australasian Conference on Computing Education, Melbourne, Australia, December, 2000.
- Schep, M, and N McNulty. "Use of Lego Mindstorm Kits in Introductory Programming Classes: A Tutorial." *Journal of Computer Sciences in Colleges* 18, no. 2 (2002): 323-327.
- Schmuck, R, and P Schmuck. *Group Processes in the Classroom*. 6th ed. Dubuque, IA: Wm C Brown, 1992.
- Schneider, G.M., and J Gersting. *An Invitation to Computer Science*. 2nd ed. 1 vols. Pacific Grove, CA: Brooks/Cole, 1999.
- Schoenfeld, A. *Mathematical Problem-solving*: Academic Press, Inc, 1985.
- Schumacher, J, D Welch, and D Raymond. "Teaching Introductory Programming, Problem-solving and Information Technology with Robots at West Point." Paper presented at the 31st ASEE/IEEE Frontiers in Education Conference, Reno,NV, Oct 10-13, 2001.
- Shin, N, D Jonassen, and S McGee. "Predictors of Well-Structured and Ill-Structured Problem-solving in an Astronomy Simulation." *Journal of Research in Science Teaching* 40, no. 1 (2003): 6-33.
- Snow, R.E., and P.L. Peterson. "Recognizing Differences in Student Aptitude." In *New Directions for Teaching and Learning: Learning Cognition and College Teaching*, edited by W.J. McKeachie, 1-23. San Francisco, CA: Jossey-Bass, 1980.
- Stein, M. "Mathematical Preparation as a Basis for Success." Paper presented at the Consortium for Computing in Small Colleges 2002.
- Thomas, L, M Ratcliffe, J Woodbury, and E Jarman. "Learning Styles and Performance in the Introductory Programming Sequence." Paper presented at the 33rd SIGCSE Technical Symposium on Computer Science Education, Feb 2002.
- Towhidnejad, M, and A Salimi. "Incorporating a Disciplined Software Development Process in to Introductory Computer Science Programming Courses: Initial Results." Paper presented at the ASEE/IEEE Frontiers in Education Conference 1996.
- Tucker, A. "Enrollments and Staffing in College Computer Science Programs: A Growth Perspective fo r1996-2000." : Dickinson College, 1998.
- Tucker, A, K Barker, A BERNAT, R Cupper, C Kelemen, and R Ungar. "Developing the Breadth-first Introductory Curriculum: Results of a Three Year Experiment." *Computer Science Education* 8, no. 1 (1998): 27-55.
- Tucker, A, and D Garnick. "A Breadth-first Introductory Curriculum in Computer Science." *Computer Science Education* 3 (1991): 272-295.
- Tucker, A, C Kelemen, and K Bruce. "Our Curriculum Has Become Math-Phobic!" Paper presented at the 32nd SIGCSE Technical Symposium on Computer Science Education, Feb 2001.

Vandenberg, S, and M Wollowski. "Introducing Computer Science Using a Breadth-first Approach and Functional Programming." Paper presented at the 31st SIGCSE Technical Symposium on Computer Science Education, Austin, TX 2000.

Vardi, M, T Finin, and T Henderson. "2001-2002 Taulbee Survey." *Computing Research News*, Mar 2003 2003, 6-13.

Wable, K. "JaCQ: An Interactive Appliation to Demonstrate Programming for CS0." *Journal of Computer Science in Colleges* 15, no. 5 (2000): 331-332.

Walker, H, S Zweben, A Tucker, Jr Myers, J, and G Braught. "The Crisis in Academic Hiring in Computer Science." Paper presented at the 13th SIGCSE Technical Symposium on Computer Science Education, Mar 1999.

Watson, G, and E Glaser. *Watson-Glaser Critical Thinking Appraisal Form S Manual*. San Antonio, TX: The Psychological Corporation, 1994.

Weiss, M.A. "Experiences Teaching Data Structures with Java." Paper presented at the SIGCSE, California, 1997.

Wickelgreen, W. *How to Solve Problems*. San Francisco, CA: W.H.Freeman, 1974.

Wilson, B, and S Shrock. "Contributing to Success in an Introductory Computer Science Course: A Study of Twelve Factors." *SIGCSE Bulletin* 33, no. 1 (2001): 184-188.

Wolz, U, and E Koffman. "Language Considerations in a Goal-Centered Approach to CS1 and 2: Java, C, or What?" *Journal of Computing in Small Colleges* 12 (1997): 12-20.

Woods, D. "An Evidence-based Strategy for Problem-solving." *Journal of Engineering Education*, Oct 2000, 443.

Yang, D, and S Wei. "A Project-Based Approach to Teaching Introductory Computer Science." Paper presented at the ASEE/IEEE Frontiers in Education Conference, San Juan, PR, Nov 10-13, 1999.

Appendix I: New Curriculum Materials

Introduction to Computer Science

Introduction to Problem Solving in Computer Science

Computer Science is a discipline that requires advanced problem solving skills in every subject area, as well as in most related tasks. Each of us is capable of solving problems in our own way, and thus a section on problem solving may seem a bit ridiculous to some people. However, each of us can work to improve our problem solving skills. Perhaps some of the things discussed here, or in this class will provide you with new tools for problem solving. Or, maybe it will confirm for you that you are using approaches that are known to work for others in the field.

Why focus on problem solving?

1. Bridge a communications gap

Those who present you with a problem have a picture in their mind as to what they think is going on. Often, their presentation of the problem fails to communicate to others what the problem is, what the boundaries are and what would be considered a "good" solution. If you are looking to become a Computer Scientist, you should expect to do a good deal of interpretation regarding the desires of others who are not familiar with terms of computing and capabilities of technology.

How would you respond to a person who asks you to "program it to make it do my work for me" or to "help me do my work"?

2. Reduce the time it takes you to find and produce valid solutions

A little thought before you act usually saves you a good deal of time later as you attempt to fix or back out of a bad solution. Failure to understand the problem fully can cause you to do a great deal of work that leads to results that weren't wanted in the first place. By the same token, it is possible that there are simpler solutions that can be used if you take the time to weigh your options before acting.

If you were given a large number of blocks (each a different size and shape) and told to stack them up to make the tallest structure you can, you could just start grabbing blocks and stacking. But, what might be a better approach?

3. Giving up is not an option

When we are faced with very difficult problems, it is tempting to give up and hope that someone else will come along and fix them. However, the more problem solving tools you have, the more likely it is that YOU can solve the problem. Problem solving becomes easier with experience. But, there are a number of things you can do to help you have success more frequently.

4. The nature of the Computer Science beast

It is the nature of most jobs and courses involving Computer Science that you will be consistently presented with problems that you are expected to find solutions for. Improving your problem solving skills can help you become more successful in the field.

5. Not just solutions - but GOOD solutions

A good problem solver doesn't just take the one solution that readily presents itself since it may not be a good, efficient, working solution. If you have ever complained about how someone else has done something (a school, business, person, etc.) and pointed out why what they chose to do was wrong, you have some idea of what you should do when you look at your own solutions. Put yourself in the position of evaluating your own solutions before using them.

6. Algorithms, diagramming and programming are all tools in the process of problem solving in Computer Science.

Five steps to Problem Solving in Computer Science

- 1. Understand and isolate the problem*
- 2. Brainstorm for ideas to solve the problem*
- 3. Design a solution that might work*
- 4. Test your solution to see if it will work*
- 5. Assess whether the solution is good enough to do it*

1. Understand and isolate the problem

If you were given these tasks - what would you do?

- write an algorithm to determine if a sandwich tastes good
- write an algorithm to produce an accounting report
- write an algorithm to grow sweet corn
- write an algorithm to judge a contest

Can you do any of these problems successfully? A better question might be, "how do you know if you've succeeded?" It is doubtful that you can answer this question since it is unlikely that you fully understand any of the problems. Thus, we have a communication issue. We need to be able to detect and identify the parts of the problem that give us vital information. By the same token, we need to detect what is missing.

A. clarify expectations

----good tasting sandwiches----

By what standard do we judge a sandwich as being tasty? Who's expectations are we meeting? What tastes good to them? How do we indicate that a sandwich is tasty or not?

B. what's the problem domain? (isolate it by identifying inputs and outputs)

----accounting report----

Which of the hundreds of accounting reports known to accountants are we supposed to produce? What

data will be used to create it? How often should we create it? What should the final product look like?

C. What are the limitations?

----grow sweet corn----

First, a problem domain question - where are we growing this corn? If it is in Minnesota, we may be limited to growing the corn in months other than November-April. Certain soils and locations may not allow corn to grow. Knowing that some things are not options can lead you to more appropriate solutions.

D. What are the rules? What is the system we work within?

----judge contest----

Yes, we still don't have a problem domain (what is the contest about?) But, even if we do, we can't successfully judge the contest until we understand the rules of the contest. What is appropriate? What constitutes success or failure in the contest?

So, if an accountant asks for reports, it is important to understand something about the accounting system the accountant works with. If a meteorologist asks to have a new system to predict the weather, it is again important that some knowledge of the rules of meteorology and existing support systems be available as a solution is being sought.

Consider this problem - a friend was working on a computer and is upset because it "isn't working"! How do you work to isolate the problem?

- clarify what "isn't working" means...what was expected to happen and what happened instead
- isolate the inputs and actions that happened just before the problem
- have some knowledge of how the software/hardware works and be aware of the limitations

2. Brainstorm possible solutions

Once you feel that you have some control over the actual identity of the problem, you can take some time to think of possible solutions. For small problems, like many you will see as examples in the text or in class, it may be difficult to see more than one solution. However, as problems become larger and more complex, there often several valid solutions. Identifying more than one possibility allows you to determine if one solution might be better than others.

Brainstorming can occur entirely in your head, or you may scribble ideas out on paper. You might have a discussion with other people about the problem. Regardless of how it happens, give yourself the opportunity to think of alternatives to solving a problem. Make yourself write options down just to get used to the idea. Good problem solving techniques require some effort. By the same token, there may not be a "best" solution, just several "reasonable" ones. Finding ways to select what appears to be the best solution for a given situation make a difficult process easier.

- There are often many possible solutions - many may be quite reasonable
- The current situation may dictate which situation is best
- Make a quick written note of possible solutions to help you remember other possibilities if one

attempt fails.

- Don't put brainstorming off!
- Let yourself think of solutions that are common as well as solutions that include new approaches and ideas
- Feel free to explore ideas and solutions that have been used in similar situations
- Not every solution is computer oriented in its entirety (even though this class focuses on that portion)
- Two heads are often better than one. Discuss ideas with other people, explaining your ideas to others often clarifies it for you.

Once you have alternatives, take some time to look at the pros and cons of each possible solution. Try to determine which one is best for the current situation.

hm. I need to wrap a present, but have no wrapping paper...what should I do? I could wrap it in facial tissues (unused). Maybe newspaper? Printouts from my last computing project? How about a paper sack? I could go get some wrapping paper.

In some cases, the solution won't be possible because the resources for that solution are not available (no printouts here). Or perhaps one solution requires more effort than the problem merits (tissue paper?). Perhaps the solution won't have the desired effect? (my spouse may not like the look of newspaper or a paper sack) Perhaps it will be worth the effort to get that wrapping paper!

3. Design a solution

Architects create models and draw up blueprints before a building is built. Electrical engineers create models and test them before they send a circuit board design out for production. Cooks follow a recipe in order to create various meals. Thus, it should be no surprise that taking some time to design a solution before actually implementing the solution saves time in Computer Science.

Give yourself permission to take the time to think ahead:

- anticipate potential problems
- look for contingencies needed if one plan doesn't work
- locate exceptions and special situations
- find borders and limitations
- work to make the solution reusable
- work to make the solution robust
- try to make the solution flexible enough to allow for changes for future needs
- develop a model that can communicate the solution to others

The design of a solution can be created using diagrams, algorithms and other models. The main purposes of creating a design that can be viewed by others is to communicate the solution. In business, you rarely work in a vacuum. Other programmers, designers, managers, etc. will need to understand what you propose to do. You may need to return to the design well after you've forgotten the details. While all of this may seem like it doesn't apply to you now, consider that the ability to develop a design with algorithms or diagrams will allow you to communicate your understanding or difficulties in understanding to teachers, tutors and peers. Sometimes the most difficult thing is knowing what to ask when you need help. Providing a diagram or algorithm of what you THINK is going on and is needed will provide others a chance to see what your thinking is.

4. Test the Solution

Once you have designed a solution you must ask yourself if it works. As a matter of fact, you should ask if it will work WELL. Take the time to trace through your design with some test information. Try more than one scenario. Try to find information that tests the borders. Choose some information that isn't supposed to work and be certain that the design handles it. Give your algorithm or diagram to another person and see if they understand it, or walk through it with them. An excellent approach in a CS class is to trade with a friend after you have both made an attempt at the algorithm. Critique each other's work and learn from different perspectives. In business, it is critical that the design is understood by others and that other opinions are solicited.

5. Assess whether you must go back or whether you can implement the solution

Is this solution really worth doing? Would another solution do better for this situation? Once you determine if this solution will provide adequate answers, you must determine if the cost of implementing it is a cost you are willing to pay. This is quite similar to the end of the Brainstorming step. Once again, you are attempting to determine if this is the best path to take. Use what was determined at that point and add to it the new knowledge you have gained in designing and testing the solution. At this point, this step should take relatively little effort since most of the work and data gathering is done!

[Problem Solving Example](#)

[Problem Solving Exercises](#)

[Return to Main Page](#)

[rfaux 8/27/00](#)

Introduction to Computer Science

Introduction to Problem Solving in Computer Science

Examples

Example 1: The French Ambassador

The French ambassador gives a reception. Half of his guests are foreigners, whose official language is not French. Each guest says "bonjour" to the ambassador. And, to be polite, each guest says hello to every other guest in the official language of the person he is talking to. The French ambassador answers "Soyez le bienvenu" to every guest. In all, 78 "bonjours" are said. How many guests are there? (1)

1. Identify and isolate the problem.

Given a problem in a test, or for a program or an exercise, it is important to first identify what is important in the question. What is important in this question?

FOCUS: How MANY guests? (doesn't include ambassador)

- 78 'bonjours' are said.... so, when do they say 'bonjour'?
- Everyone says 'bonjour' to the ambassador
- The French ambassador never says 'bonjour' - how do I know that?
- Half the guests are French speakers (because half are NOT)
- Each of the French guests are greeted with 'bonjour' - how did I determine this?

2. Brainstorm - how might I solve this?

Think of some ways that you could work to solve the problem. Often, this step, for this type of program can be very short. However, a few moments here, can save lots of time later!

- Pick numbers and see if they work
- Develop an equation and solve for an answer
- Come up with a logical sequence/algorithm that leads to the solution

Which one seems to be easiest from your perspective?

- It could take a very long time trying numbers before I find the right one, but I don't have to think much in picking a number.
- If the equation is correct, I'll get right to the answer. But, it is difficult to find an equation...unless.....
- an algorithm may be a combination of educated guesses and equations. But, it may be difficult to see a step by step solution!

3. Design a way to get the solution

Lets try the third method. We'll try to combine logic and some educated guesses!

1. Assume a variable for the number of French guests (x)
2. Figure out how many 'bonjours' are said to one of the French guests
3. Figure out how many 'bonjours' are said to all French guests
4. Figure out how many of the 'bonjours' were said to the ambassador
5. Add up all of the bonjours in 3 and 4 and it should be 78 'bonjours'
6. solve the equation that results for the number of guests.

4. Try it out!

There were 78 bonjours and x guests.

Every French guest was greeted by a 'bonjour' from every other guest. That means a French guest was greeted by $2x - 1$ 'bonjours' (they won't greet themselves!)

So...each of them were greeted by 'bonjour' $2x-1$ times. So the total number of bonjours to French guests is $(x)(2x-1)$

the ambassador was greeted with 'bonjour' by every guest...which means he heard 'bonjour' $2x$ times (because there are $2x$ guests!).

So... $(x)(2x-1) + 2x = 78$

solving, we get 6 for x . So there were 6 French guests and 6 non-French guests.

12 total!

Note: Problem solving often entails practice and experience. Once you see an approach to solve a certain problem, you can look to use that approach again. However, many problems don't fit patterns we recognize. So, the more problems you work to solve, the more patterns you have to work from in your own experience!

Example 2: What's missing?

Problem: Water the plants before they die.

This seems to be a simple problem to solve on the surface. But, what information do you need in order to actually be successful at it? After all, you could continuously water the plants, which would certainly meet the requirements of the problem as stated!

In Computer Science, we are often required to learn more about the subject and to extract information from those who are requesting the service. In this case, you might want to know some of the following:

- What types of plants are they?
- How much water do they need when I water them?

- How often should I water them to keep them from dying?

But, perhaps, even the problem statement isn't entirely adequate. Maybe what we really want is to encourage the plants to live and grow! In that case, perhaps we need to know more beyond how much we should water, such as:

- What kind of light do these plants need?
 - What do I do if they are infested by bugs?
 - Should I trim off dead leaves?
 - Should I pour the coffee sludge from my cup into the potted plant?
-

[Problem Solving Notes](#)

[Problem Solving Exercises](#)

[Return to Main Page](#)

[rfaux 8/31/00](#)

Introduction to Computer Science

Introduction to Problem Solving in Computer Science

Exercises

1. For each of the following problem statements, identify things you need to know to solve the problem, but aren't given in the problem statement.

- a. Compute charges for the telephone company's bill.
- b. Invest money in a stock that will increase significantly in value.
- c. Determine the average test scores for students in a class.
- d. Find the smallest number in a set of numbers.
- e. Compute the take home pay for employees in a business.

2. For the following problem, identify the parts of the problem that are critical to solving the problem. After you have done that, is there something more that you must know in order to solve the problem? In other words, must you make an assumption or get a clarification in order to do this?(hint: yes there is! What is it?)

Timothy goes to a fountain which delivers an unlimited amount of water. He brings two empty containers, one of 7 liters, the other of 11 liters. How many operations does he need to fill one of the containers with exactly 6 liters of water?

3. Go through the steps of problem solving and attempt to develop an algorithm to find the total, average and largest number in a given list of 25 numbers.

[Problem Solving Notes](#)

[Problem Solving Examples](#)

[Return to Main Page](#)

[rfaux](#) 8/31/00

Introduction to Computer Science

Flow Charts

I. Why Flow Charting?

Often the best way to visualize a problem or a possible solution is to draw pictures or representative models of the things we are working with. Pictures can provide us with a different perspective and help us to see relationships between objects and actions. Pictures often provide us with a more complete idea of the situation than a series of short word phrases can. However, pictures combined with text provide an extremely powerful tool for communication and problem solving.

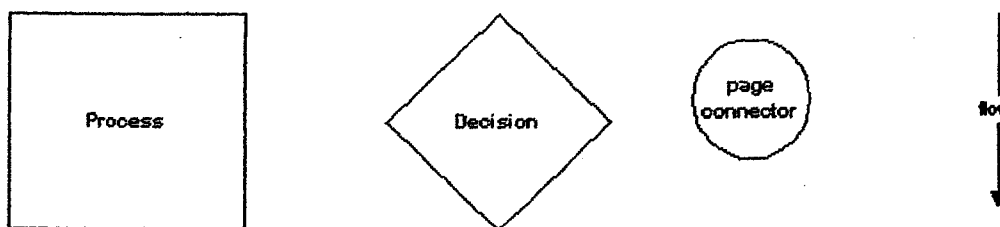
Some individuals find that they can develop algorithms more quickly if they utilize a diagramming technique, such as flow charting to represent their algorithm. In many cases, people are able to work through an algorithm utilizing a technique such as flow charts and then convert the flow chart into pseudocode. In other cases, a person works best with pseudocode.

In either case, good pseudocode and good flow charts are two roughly equivalent tools for representing solutions. Pseudocode is quite often capable of handling higher amounts of detail and (obviously) greater text content. Flow charts are usually a better tool when an algorithm needs to be shared with other persons since it requires less effort to understand.

II. Control Structures and Flow Charting

Flow charts are just one way of using diagrams to illustrate problems and possible solutions. This set of symbols is the most frequently used set. However, there are other symbol sets used for the same diagramming purpose. Understanding of one style of flow charting makes it easier to understand any other style of flowcharting.

Flow Chart Symbols



square - process or module or action

diamond - decision / check a condition and branch to proper process (square) or another decision (diamond)

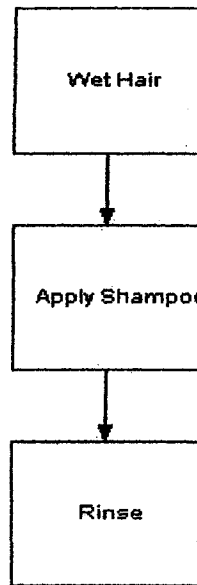
circle - indicates that the diagram continues on another page (label)

the symbol to show where to go)

lines - indicate the flow of the system from one process or decision to another

Sequential Structure

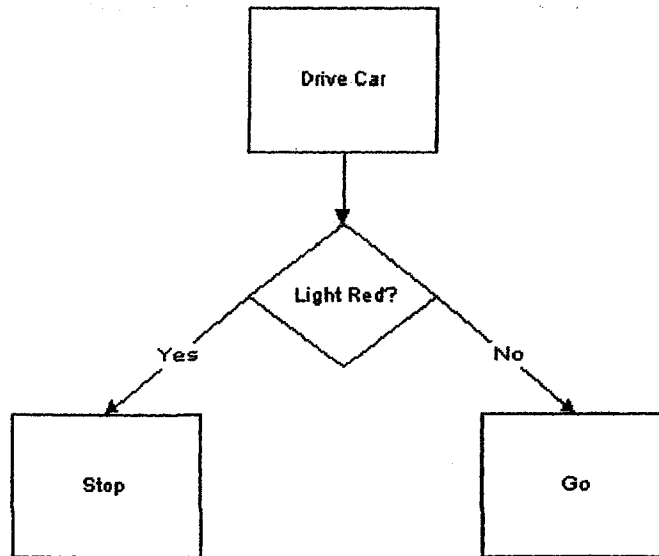
A series of processes that follow in order.



Selection/Condition Structure

A condition exists that may change the order or types of processes to be followed. Often referred to as an IF/THEN situation.

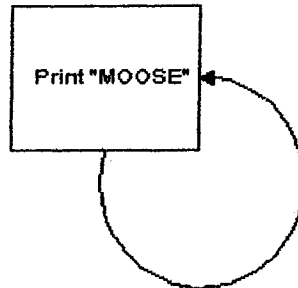
For example, IF the light is red THEN I will stop OTHERWISE I will go.



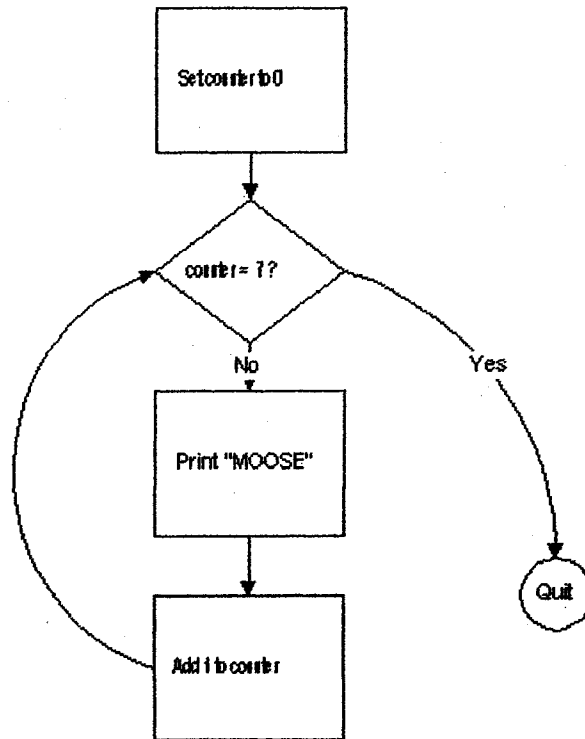
Loop Structures

Often, we might wish to perform the same set of processes a number of times, we can perform a loop and do the same set of actions over and over until a STOPPING condition occurs.

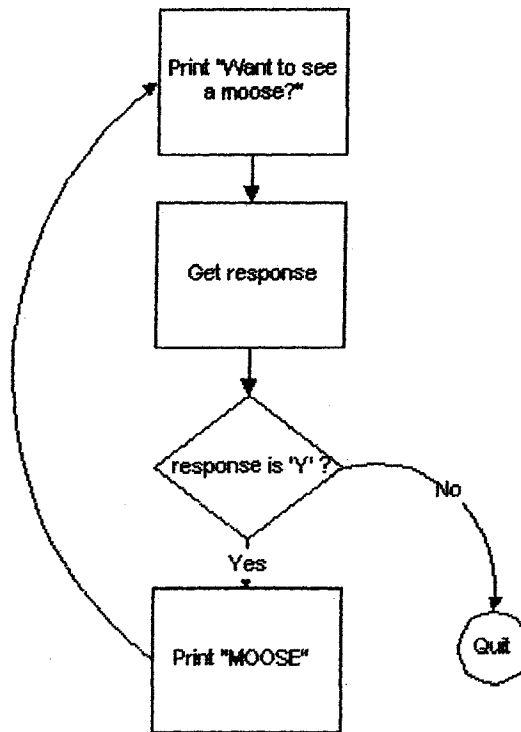
Failure to provide a STOP condition will cause the process to go into an INFINITE LOOP



An example of a FOR LOOP could be to display the word 'MOOSE' on the screen 7 times.



An example of a WHILE LOOP could be to display the word 'MOOSE' and then ask the user if they'd like to see it again. If they say "NO" quit displaying moose.



III. Using Flow Charting to Solve Problems

Flow charting provides some of the same tools that pseudocode brings for algorithm development. The main difference is that pictures convey a great deal of the solution. For many people, a diagram can communicate a possible solution far better than a series of words. For others, drawing a quick picture helps a person to brainstorm far more than a set of pseudocode.

Hints for using flow charts:

- Don't worry about neatness when you are brainstorming - just get your ideas down on paper.
- Use pencil so you can erase and try new ideas.
- Take the time to walk through your algorithm to see if it actually does everything you think it should do
- Don't assume that something 'magically' happens. With computers you have to tell it everything - thus your flow chart should have all of the steps!
- Expect your first draft to have problems!
- Consider finding a problem with your algorithm a success, not a failure (a failure is when a person using your code finds a problem you missed! So YOU should work to find errors)

[Flow Chart Example](#)

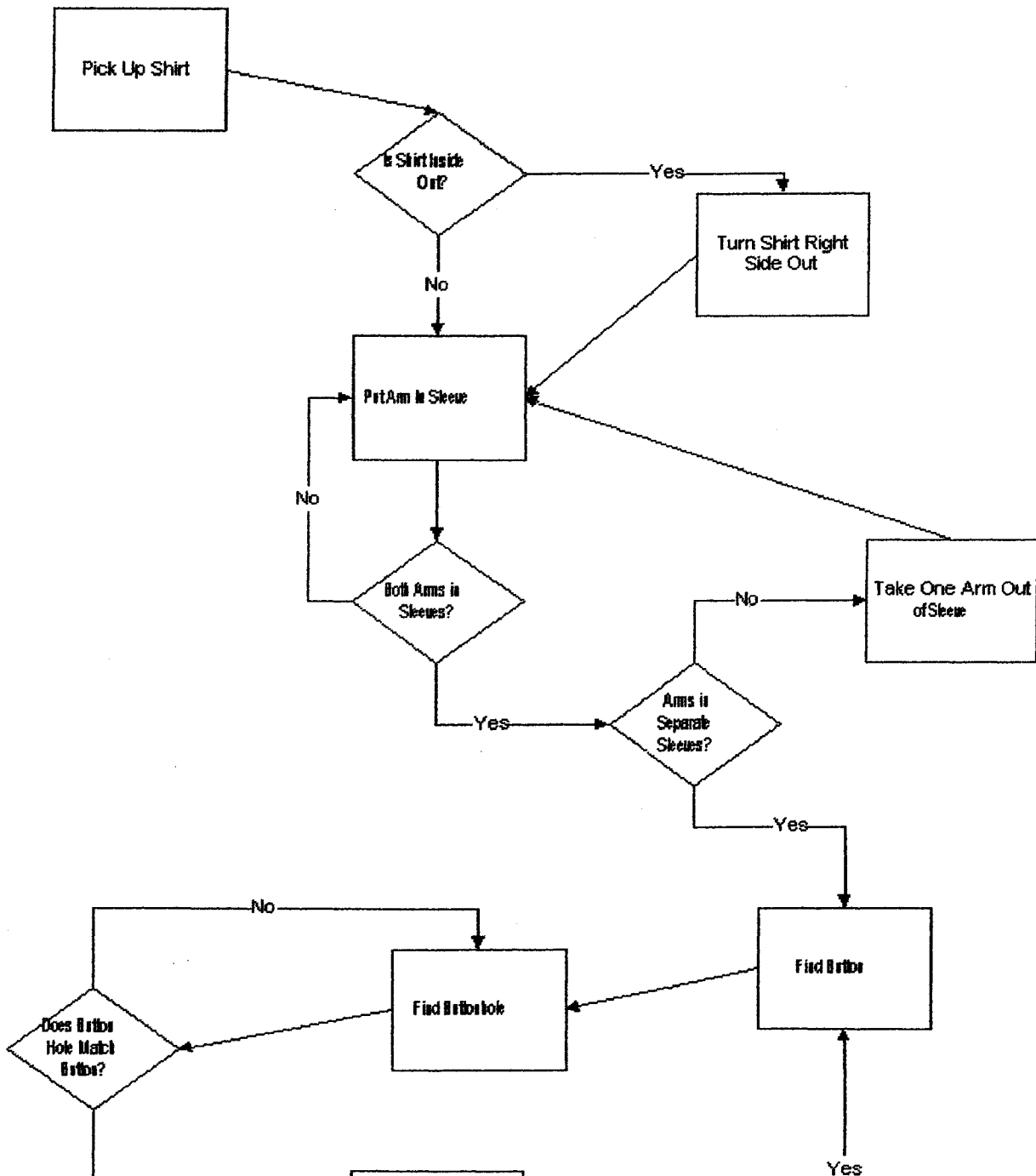
[Flow Chart Exercises](#)

[Return to Main Page](#)

Introduction to Computer Science

Flow Chart Example

The following example is an attempt to provide an algorithm for putting on a shirt. Consider the possibility of attempting to tell a robot (which must have everything given to it) how to do this:



Identifying structures:

- Sequential structures are indicated by an arrow showing direction but no decision (yes/no)
- Conditional structures are shown by diamonds
- Loop structures can be found in multiple places - always with a conditional structure to allow escape from the loop

Take a few moments and identify every loop in the flow chart.

Testing the Algorithm:

It may seem a little odd to have to ask if ones arms are in separate sleeves. However, consider the robot and its lack of knowledge pertaining to shirts. We might have to go so far as to define shirt, sleeve, arm, button, buttonhold (to name a few). As humans, we tend to have experience in recognizing these items, so the questions are fairly simple to us.

Are there ways that this algorithm can fail?

- Both arms could be in different sleeves, but the **WRONG** sleeve (left/right)

Do you see other potential problems?

[Flow Chart Notes](#)

[Flow Chart Exercises](#)

[Return to Main Page](#)

[rfaux](#) 9/15/00

Introduction to Computer Science

Flow Chart Exercises

- For each of the problems below, develop a flow chart and a segment of pseudocode.
 - Alternate between these two methods by doing the flow chart first on one problem, then write the pseudocode for that problem. For the next problem, start with a pseudocode solution, then go to the flow chart.
 - Once you have completed a couple of problems, take time to have another learner review your work while you review theirs.
 - Remember - small details are important here!
-

1. Count to 100
 2. Count to 100 by 2's
 3. Count to 100 by 3's
 4. Add the numbers from 1 to 100
 5. Add up 10 numbers that are given to you - that is, you don't know what they are until someone gives them to you....
 6. Find the average of 10 numbers given to you
 7. Find the average, high number, low number, sum of those 10 numbers
 8. Display three numbers in order from smallest to largest (the numbers are given to you)
 9. Display three words in order (the words are given to you)
 10. Display 10 numbers in order
 11. Display 10 words in order
 12. Find the batting average of your favorite baseball/softball player
 13. Display 10 baseball/softball players in order of their batting average (high to low)
 14. Find the area of a rectangle (two sides are given to you)
 15. Find the area of a triangle
 16. Find the area of a circle
 17. Find the area of a snowman (3 circles - we'll stay 2 dimensional here!)
 18. Find the area of a house (a rectangle and a triangle)
 19. Find the area of a rectangle with a triangular hole in it
 20. Modify your area solutions to check for values given to you that are less than zero and ask for a new number
 21. Ask a person for a number between 1 and 100, ask again if they give you a number outside that range
 22. Swap the contents of two glasses of water - what will you need to do?
 23. Calculate a student's grade point average - $A = 4$, $B=3$, $C=2$ and $D = 1$
 24. Find the factorial of a set of numbers (multiply from 1 to n)
 25. Find the sum of a set of numbers (add from 1 to n)
-

[Flow Chart Notes](#)

[Flow Chart Examples](#)

[Return to Main Page](#)

[rfaux 9/14/00](#)

Introduction to Computer Science

Testing Algorithms

1. Successful Testing

Some axioms for testing:

1. A successful test is one that finds a problem.
2. The most successful tests are those that find problems early in the process.
3. You can't catch them all, but you sure can try!
4. Test and test again.

1.1 A successful test is one that finds a problem

This concept is a difficult one to accept. After all, it is our goal to find a solution that works, not one that breaks! So, it is normal when people get upset when a test shows us that a solution didn't work. The natural result is to think that the test failed. In reality, the test succeeded in catching a problem with the solution. The test was successful in preventing you from submitting a solution to your instructor, to your boss, or to your customers with a flaw.

Consider this: How many times have you used a computer or piece of software and gotten upset when the computer or software crashes on you? Who do you blame? Who should you blame? As a consumer, wouldn't you like to think that those creating the software are taking the time to thoroughly test their systems before putting them into action? If that's the case, perhaps you should consider that you are looking to be a part of the software development process and that you will be one of those responsible for either successfully ferreting out problems before they reach the customer OR for allowing them to reach the customer!

1.2 The most successful tests are those that find problems early

In business, the cost of fixing problems goes up as you progress through the process of developing software.

Nearly all software development models agree that there are three major phases:

1. Definition
2. Development
3. Maintenance

As we progress through these phases, the cost of change increases greatly.

- Development - 1.5 to 6 times as much
- Maintenance - 60 to 100 times as much

But where do these costs come from? Or are they just numbers intended to try to scare us into doing something we don't want to do in the first place?

Project this onto doing homework for college courses. Assume you are working on a paper for a writing class. You sit down at the computer, fire up the word processor and begin typing. You type in four pages of material. You print it. You proof read it. You edit it. You bring it in to hand to the teacher. The teacher informs you when you get it back that you wrote on an incorrect topic and in an incorrect format! The cost is now either a very bad grade or a lot of wasted time as you redo the entire project. Wouldn't it have cost you less to have reviewed the assignment and asked questions and clarified the project earlier? Perhaps if you'd written an outline and then discussed it with another student, or an instructor, or a tutor, you could have caught this well before this point. Wouldn't that have cost you less in the long run? This process is even more important when one develops software.

Take some time to think of reasons why the cost is so much greater in software development in later phases of development. Think in terms of time, money, effort and other resources for the business.

1.3 You can't catch them all, but you can sure try!

Even things that seem very simple can become very complex when you attempt to implement the idea with a computer. Lots can go wrong. For example, consider a program that is created to calculate and select appropriate change in terms of hard currency denominations (\$5, \$1, quarters, etc). Seems simple enough when you do it yourself, but try telling a computer how to do it with a piece of software! Check out the exercise linked at the bottom of this page.

The idea here is that you can not expect to catch every problem. Programs are complex enough by themselves. But, when you add to it the possibility that it may react negatively to another program, such as certain operating systems, it is impossible to catch them all. However, we must remember that the customer is relying on the software product. It is necessary to be able to isolate and remove as many problems as possible. Similarly, it is necessary to continue to watch for errors as things change! (see the next point!)

A final point, we can't delay software products forever! They need to be released for use at some point in time. This is where software developers need to consider the level of reliability necessary for the product - which is considered in the test plan for that product. For example, if the product works in a life/death situation, it had better be extremely reliable.

1.4 Test and test again

Every time the solution design is modified, there is a chance that something that worked before will break. Do not assume that your new modifications have no effect on things that were fine the last time you looked. Test each time you make a change. Remember that once you perform a certain test, it is easier to perform a second time since you already know how to set the test and make it go!

2. Vocabulary in Testing

Black Box Tests - A test where the internal workings are not known. Something is put in and a result (correct or incorrect) comes out.

White Box Tests - A test where the internal workings are clearly seen. Something is put in and we can watch the result be generated.

Integration Tests - Making sure a part of a product works with other parts of a product (or with other products).

Boundary Testing - Providing test inputs that check values that are on one side or the other of getting different results.

Walkthrough - Any method of reviewing a solution for validity. Tracing is one type of walkthrough. Often, walkthroughs are done with other team members.

Tracing - taking an algorithm and stepping through the solution with input to see how it formulates a result - a white box type of test!

Test Plan - A strategy for efficiently and effectively testing an algorithm, solution, program or product.

Test Suite - a series of tests available to run through an algorithm, program or product.

Requirements Comparison - A test plan should include testing that reflects the requirements for the algorithm. In other words, the system should do what it is required to do. No more, no less.

Exhaustive Testing - making sure every possible path through the algorithm is checked for correctness.

3. A Basic Testing Strategy

As you progress through the CS program you will find that you will learn and use more strategies in testing. However, a good starting set of methods will help you get through this class and the early programming courses. The following is a strategy that is effective for early algorithm development and program coding.

- **Trace** your algorithm with various inputs to make sure it does what the requirements (*requirements comparison*) tell you it should do. When you trace, you should keep the next two in mind.
- Make sure you exercise every line of the algorithm or every box/diamond of the flow chart (*exhaustive testing*) by selecting different tests that go to each part. (note: if you can't GET TO a part of your algorithm - maybe it is not needed - or you made a mistake in design)
- **Test boundaries**, they often show you where you MISSED something in your design - so you might have to add to the solution. (example - test negative numbers, zero, etc.)
- If your algorithm is large, you should keep track of the tests you run and their results. The bigger the algorithm, the more likely it is that you should develop a *plan for your testing*.

4. Avoiding Pitfalls, Burnout and the Occasional Dragon

Test smart and avoid the 'brute force' mentality. We may admire someone who has the desire and ability to work themselves to exhaustion as they test every possible input. One example of testing

smart:

- The problem is to print grades for a test. As follows

A	90 and up
B	80 - 89
C	70 - 79
D	60 - 69
F	less than 60

It makes sense to test data for each grade. It also makes sense to test boundary input, such as 90 and 89. But, why would you test every number from 80 to 89? Is there something in the solution that will be different for 85 and 86? As an exercise, try to develop an algorithm for this and test it.

Test consistently and intelligently. Don't put it off. Instead, do a little bit as you develop the design. As you practice this, you will find that testing becomes easier and that you get better at selecting tests and at implementing them very quickly.

5. Consequences of Failed or Inadequate Testing

As mentioned before, the cost increases when a problem is discovered late in the development cycle. Part of this has to do with the increased difficulty in altering the solution to handle the problem. Simply remember that band-aids do not stick as well as your own skin does to the rest of your body. Something that is designed into a solution from the beginning usually is more seamless and less likely to fall apart than something that is done to modify an existing solution just to 'get it to work.'

Once software is created, it is subjected to an aging process known as the bathtub curve. Early in its life, the software is prone to breaking down as errors are discovered and removed. It then has a period of relative stability until changing requirements cause it to begin breaking more frequently. The more fixes that occur later in life for the software, the more the software 'ages' and becomes more prone to break. Catching errors in design reduces the number of weak points in the actual software! Thus, a longer life is possible for the product you design!

Some examples of real world consequences when testing was not sufficient:

- online auctions - database query problems cause auction lots to disappear during the last hour of the auction's activity. The last hour is frequently when the most bids are made on items. The result - sellers (who pay the online auction company to list items) receive significantly lower dollar amounts for their items.
- military - anti-missile weapons trained to site on missile 'flame trails' notice the Hale-Bop comet and target it. If they hadn't been stopped, what would have been the result?
- medical - radiation treatment machine overdoses patients. The result, burns, infections, severe pain and, in some cases, death.
- manufacturing - a chip insertion machine crushing microchips by attempting to insert them in the wrong position on a circuit board. The result, an entire set of chip production for a new product is ruined.

Introduction to Computer Science

Testing Algorithms - Examples

Example 1: Division

Algorithm:

1. Input A from Keyboard
2. Input B from Keyboard
3. Set C to A divided by B
4. Display C on screen

Trace code with sample input:

- | | |
|----------------------------|-----------------------|
| 1. Input A from Keyboard | A is 8 |
| 2. Input B from Keyboard | B is 4 |
| 3. Set C to A divided by B | C is 2 |
| 4. Display C on screen | <i>2 is displayed</i> |

No problems found with a general test.

Try B as a zero!

Trace code with simple input:

- | | |
|----------------------------|--------------|
| 1. Input A from Keyboard | A is 8 |
| 2. Input B from Keyboard | B is 0 |
| 3. Set C to A divided by B | C is ??????? |
| 4. Display C on screen | ????? |

We can't divide by zero, so we should prevent a zero from being entered for B. This was a successful test since we found a problem.

Modify Algorithm:

1. Input A from Keyboard
2. Input B from Keyboard
3. If B = 0 Go to 2
4. Set C to A divided by B
5. Display C on screen

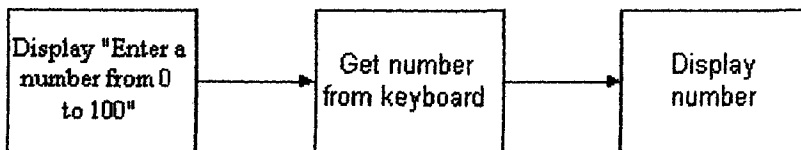
- | | |
|----------------------------|-----------------------|
| 1. Input A from Keyboard | A is 8 |
| 2. Input B from Keyboard | B is 0 |
| 3. If B=0 go to 2 | true |
| 2. Input B from Keyboard | B is 2 |
| 3. If B=0 go to 2 | false |
| 4. Set C to A divided by B | C is 4 |
| 5. Display C on screen | 4 is displayed |

Another consideration with this example. What would the computer user see on the screen with this algorithm? Anything? the only thing with this algorithm displayed on the screen is the number 4. How do they know they are supposed to enter numbers via the keyboard? Often, in early design, such things are not a concern, but they should be considered for the final product.

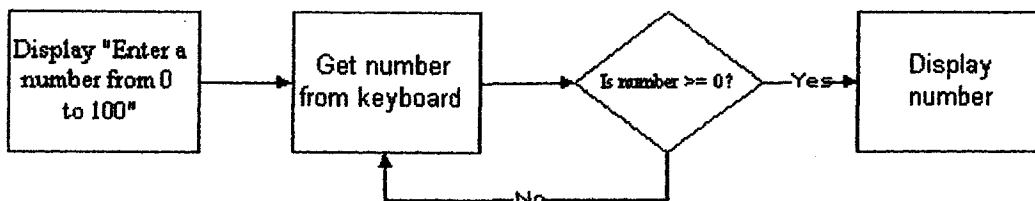
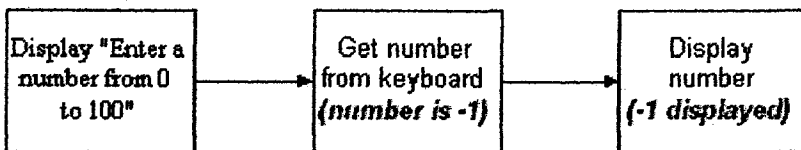
Second modification:

1. Display " Enter two numbers to be divided" on screen
2. Input A from Keyboard
3. Input B from Keyboard
4. If B = 0
 - 4.1 Display "We can't divide by 0, enter a new second number"
 - 4.2 Go back to 3
5. Set C to A divided by B
6. Display "Your Answer is"
7. Display C on screen

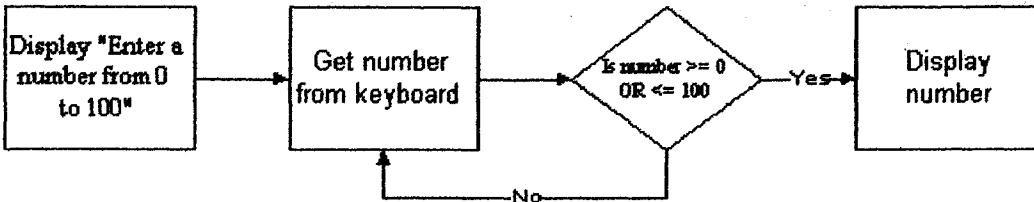
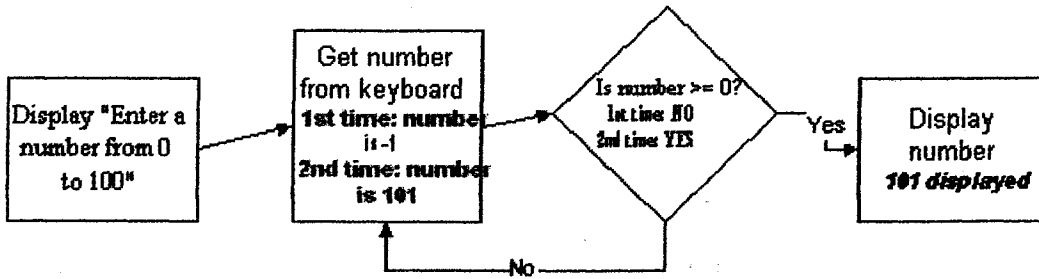
Example 2: Get a number between 0 and 100



Test: What if number entered is -1 ? What happens? Use a pen or pencil and draw in values to help yourself trace!



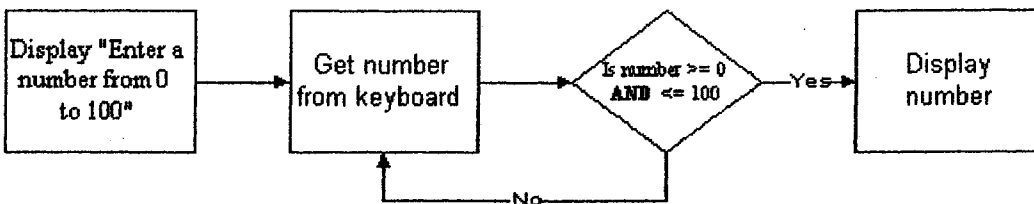
Test: Does -1 work now?
Test: How about 101?



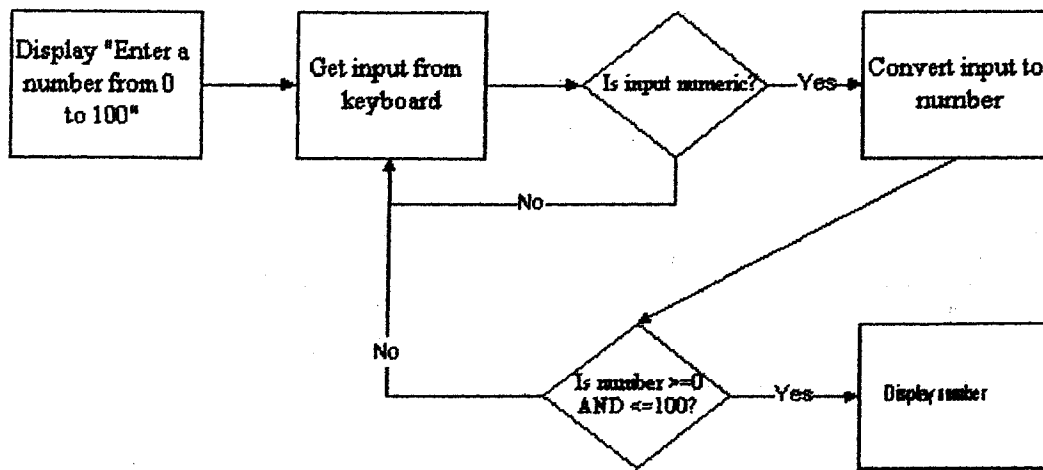
Test: Now try -1. Remember, changes can cause things to break!

Tracing is left to you this time!

-1 fails to work. The current condition will allow ANY number to succeed because any number entered will be either bigger than 0 OR smaller than 100! Maybe we should change things a little?



Test: Does -1 work now?
Test: How about 101?
Test: What if I hit the letter "a" ?



Here is a process that will allow us to handle a non-number entry! Don't assume that clients who will use your solutions will avoid these errors!

[Testing Notes](#)

[Testing Exercises](#)

[Return to Main Page](#)

rfaux 9/25/00

Introduction to Computer Science

Testing Algorithms - Exercises

Exercise 1: The change making machine

Assume that you have to create an algorithm that will give change if you are given the inputs:

- Cost of item
- Amount given for payment
- number of bills and change in machine

Assume we won't have amounts over \$10.

Assume we start with the following:

\$10 bills	0
\$5 bills	2
\$1 bills	5
25 ct coins	10
10 ct coins	10
5 ct coins	10

Start with this algorithm - then try some of the tests given below and modify the algorithm if you feel it needs to be modified after you try the test.

1. Get amount due
2. Get amount paid
3. Change = amount paid - amount due
4. While Change > 0
 - 4.1 if Change > 5
 - 4.1.1 Give a \$5 bill in change to customer
 - 4.1.2 Subtract 5 from Change
 - 4.1.3 go back to 4
 - 4.2 if Change > 1
 - 4.2.1 Give a \$1 bill in change to customer
 - 4.2.2 Subtract 1 from Change
 - 4.2.3 go back to 4
 - 4.3 if Change > .25
 - 4.3.1 Give a quarter in change to customer
 - 4.3.2 Subtract .25 from Change
 - 4.3.3 go back to 4

- 4.4 if Change > .10
 - 4.4.1 Give a dime in change to customer
 - 4.4.2 Subtract .10 from Change
 - 4.4.3 go back to 4
- 4.5 if Change > .05
 - 4.5.1 Give a nickel in change to customer
 - 4.5.2 Subtract .05 from Change
 - 4.5.3 go back to 4
- 5. Say thank you.

This algorithm looks pretty good at first glance. But, there are problems. You may see some now if you look. But, try some of these tests and see what you find.

- A. Try an amount due that is larger than the amount paid.
- B. Try an amount due that is larger than \$10....can we do that? Does our algorithm prevent it?
- C. Try a negative amount paid.
- D. Try an amount due of \$0
- E. What happens when a second customer comes in? Are the amounts of bills/coins accurate now?
- F. What happens if you run out of \$5 bills?

Can you think of other successful tests that find problems with this algorithm?

Exercise 2: Grade determination

- The problem is to print grades for a test. As follows

A	90 and up
B	80 - 89
C	70 - 79
D	60 - 69
F	less than 60

It makes sense to test data for each grade. It also makes sense to test boundary input, such as 90 and 89. But, why would you test every number from 80 to 89? Is there something in the solution that will be different for 85 and 86? *As an exercise, try to develop an algorithm for this and test it. As a further exercise, develop the algorithm first with either flow charts or pseudocode. Then, rewrite it in the other format.*

Testing Notes

Testing Examples

Introduction to Computer Science

Team Development

I. Teams in Computing

Software development teams take many forms in the workplace. Some of these incarnations work well, and others do not work at all. The following roles are found frequently in IT environments, though the roles may not be labeled or recognized by the company or the members of the team.

- **Project Leader** - this individual is primarily responsible for the overall vision and direction of the project. Often the project leader is also an analyst (and has something to do with high level design) and may be the client group contact. Some businesses equate the project leader with an administrative position and is filled by a manager or supervisor. Others believe that there should be a separate project leader from the administrative leader.
- **Lieutenant(s)** - this individual or individuals frequently consult with the project leader on the overall strategy for the project's development. If the project is large enough, each lieutenant will be responsible for a portion of the required duties or functions. Thus, they become project leaders for smaller portions of the overall project. These persons often serve as analysts, documentarians, client group contacts and programmers.
- **Documentarian** - this person is ultimately responsible for tracking all requirements, design, decision making documents. They do not necessarily create all such documents, however, they will serve as the librarian in order to keep track of current versions of materials. This person often serves other duties in the group as well (except for very large projects).
- **Client Group Contact** - the person primarily responsible for answering questions raised by the clients and discussing requirement and design options with them. This person should have excellent communication skills and should be able to learn the skills and information necessary to perform the tasks of the client group.
- **Analysts** - persons responsible for analysis and design of the product. Can include the process from requirements brainstorming to detailed design. Often, companies employ persons as programmer/analysts. Thus, they switch to coding once the design is completed.
- **Programmers** - in larger projects, these persons are responsible for converting design into working code. These people are more concerned with dealing with syntax and system issues and will often refer to the analysts if design issues arise.
- **Test Engineers** - some companies use a subset of the client group to test the product. In other cases, specially trained individuals design and implement test suites for the product. These persons are often involved starting at the design stage. Often work in conjunction with analysts and programmers.
- **Specialists** - other individuals may be needed for the project. Client group experts, Data Base Administrators (DBA), technical support personnel, and others may be called upon to be a part of such a team.

What determines the population of a project team?

The **ideal** situation is to let the complexity of the project dictate the population and distribution of the team members. A small, simple project may consist of a single programmer/analyst who reports to a project manager. A very large, complex project may have a directing project leader with a number of lieutenants who serve practically as project leaders for each component of the project.

The **reality** is that the existence or lack of managerial and organizational support for teams will determine the composition of the teams themselves. In some cases, teams are simply organizational units that are expected to work on multiple projects at one time. No other organization is enforced or encouraged.

Factors that lead to success or failure in these teams:

6- project length of 6-9 months is usually considered to be optimal for larger development. Longer projects lead to development problems.

5- Too many members in the team leads to too little progress because of overlapping tasks and team conflict....too few members leads to overwork and delayed progress.

4- The ability to determine which tasks are critical tasks is important. A critical task is one that must be completed in order for another task in the project to even begin.

3- Lack of documentation leads to teams going in circles as they rediscuss and redecide issues!

2- Pressure to code the product before the design is ready leads to product delays and errors.

1- Communication among team members and with management and the client group is critical for success. Good communication can overcome many of the other factors listed here when it is combined with a team that works together for success.

II. Building a Working Team in CS classes

General Goals for a CS team:

1. Strive for continuous improvement versus delayed perfection!
2. Avoid the Hero Syndrome
3. Pay now, instead of later.
4. Paper trails - being followed can be a good thing.
5. To collaborate, you must communicate

Explanations for these follow:

Do not procrastinate because you are waiting for everything to be perfect before you seek to present what you have done to the rest of your team. Collaboration with team members means you should learn to accept both help and criticism from each other. Also, as an individual, seek to do some work on a project daily, rather than trying to do it all in 'one big push' on the day before it is due! We all know this at one level or another, but it doesn't hurt to remind yourself!

No one member in the group should feel that they have to provide inhuman amounts of effort and work in order for the project to reach completion. At teams, each member should play the hero. But, no one person should be required to carry the load all of the time.

If there is a problem between team members, with team organization, or with the project itself, it is wiser to deal with the problem as soon as it is known. Allowing problems to remain unaddressed only gives individuals time to get more upset about the situation.

Documentation of team decisions is a preventative against individuals changing the product as they desire. In a team development situation, each individual needs to work within the team organization in order to promote the possibility that all the parts will work together in the end! This does NOT mean that ideas that are different should be ignored. But, once an agreement is reached and documented, it shouldn't be changed unless the group agrees that the decision is no longer correct!

And, finally, if you want to be a successful group, you must work to improve individual and group communication skills!

Setting Project Goals for a team

The obvious goal for any project is to complete the project. However, how do you get to that goal?

One good way to start is to begin with team organization goals:

1. Determine how often you will meet and set a schedule for general meetings during the project duration. Such things can always be changed if the team decides it is needed. However, if each member agrees to and knows that meetings every Wednesday at 4:30pm will be the norm, it gives them time frames of preparation for work with the team.
2. Determine how often the team will expect individuals or committees to produce something new for the project. A wise choice would be to expect each participant to have something new to report at each scheduled weekly meeting. This promotes continuous progress
3. Determine how you will keep track of what you do and what your decisions are. This may require that you get someone to volunteer to be the documentarian for your group. this person may take notes at meetings and will keep all current versions of work delivered by team members each week. Note: this person should not be expected to type out all documents. Instead, they are a librarian and should know what documents they have and be able to locate items quickly when a question needs an answer.
4. Determine which team members will take on what roles in the team. For most teams, members have definite preferences for their own role. Volunteering and discussion of needs usually leads to a good distribution of work. One individual should be in charge of keeping an eye on the even distribution of work and should call attention to situations that might point to the 'hero syndrome!'
5. Break down the project time-wise. Start at the end date and work backwards to the current date. Come up with a tentative plan for where you will be each week!
6. Break down the project task-wise. Try to take the large task and break it down into smaller tasks that can then be distributed to team members. Remember that some tasks won't be known until later in the process.

As the project continues, you may make changes to earlier decisions and you will find new tasks and needs. But, hopefully, your early organization will give you the tools to handle them.

III. Communication

A. Work on you LISTENING skills

Listening to others is not easy. It *requires practice, patience and endurance*. Most people have very little concentration when it comes to listening. And yet, we all seem to expect that others will listen carefully to us when we speak.

Consider this: in your last conversation, how much of the other person's comments do you remember? How much of your own opinions and comments do you remember? How much did you speak? How much did they speak? Did anything anyone else say make you stop and think about something you hadn't already thought about? Or did something they say make you think about something in a different way?

Odds are you remember more of what you said than what they said. It is also likely that you discarded much of what was said to you if you didn't already agree with it. It is also likely you couldn't even accurately describe what the other person said to someone else. *Work on your endurance in communication by reminding yourself to listen carefully* - expend the energy and consider another person's thoughts. You will maintain the right to disagree even after you listen!

Be an active listener - show you understand what is being said by rephrasing their comments in your own words. Ask if what you said matches what they are trying to tell you.

Instead of pretending to understand something that you don't, but willing to *ask for clarification*. It may be mildly embarrassing now...but imagine how embarrassing it will be if you still don't understand a week later when you are asked to give a report on how you did on the thing you didn't understand!

Prepare yourself to listen and to learn when others are talking. Take a second and give yourself a mental "kick" to be an active listener. Allow people to finish their thoughts before you start yours. Have some paper available to you so you can jot notes regarding thoughts you have as others think. That way you can mention these later if you think they will be useful later....without interrupting the speaker.

Finally, *reading someone else's work is an equivalent to listening*. Much of the communication between team members is written. Failure to carefully read and analyze other member's work can lead to crippling miscommunications for the group. Take the time to read carefully and put notes in the margins as you read. Ask questions if things are unclear and be critical, but constructive with your comments.

B. Make Yourself CLEAR

Speaking well is very important if you wish to make your ideas clear to others in the group. Computer Science is no longer a field that should be populated with persons who are unwilling or unable to communicate.

Some speaking hints:

- Try to make your point early, then support that point
- Watch for non-verbals that tell you the other person doesn't understand or isn't listening
- Allow listeners to ask questions periodically
- Allow listeners to voice their opinions and give them the respect of listening you hope they will

- give you.
- Be patient. Expect to explain things more than once.

Writing well is also very important. Computer Scientists are expected to write concise descriptions that explain designs, tests and products. Those who write poorly look less skilled than those who do not. It does not always mean that the better writer is a better designer or programmer, but their writing skills make it easier for them to communicate their ideas and skills to others.

Some writing hints:

- Write an outline first.
- Proofread what you write.
- Use spell checkers and grammar checkers - but use them wisely.
- Ask someone else to proofread (and proofread for them in return)

Make the improvement of your own ability to speak and write a goal for yourself. The jobs most coveted in Information Systems fields are won by those with superior communication skills.

C. Documentation

The paper trail created by documentation can prevent a project group from being crippled by various events. Consider these scenarios:

- One member of the group has personal opinions that rarely agree with everyone else.
- Someone is no longer a member of the group, or is absent during some meetings.
- A new person is added to the project or replaces another individual
- Management (or the teacher) asks for evidence of progress of your project

Documentation serves as a shield against those who have personal agendas. In one instance, an individual did not like a decision reached on a topic a week ago. They attempted to change the decision by introducing the same topic two weeks later. However, the documentation with the decision was used to inform all members that this topic was no longer valid and had been decided. Otherwise, the group may have been stuck discussing the same topic for another three hours.

Documentation serves as a reference when someone is absent or a new person joins the task force. Good documentation can prevent the absence of an individual from crippling the group entirely.

Documentation serves as a promotional tool to show progress and needs to management, client groups or instructors.

IV. Rules of Conflict

A. The Role of Dissent

Is the world flat? If it weren't for dissenting opinions, people might have continued to think this for some time. It is important to recognize that the willingness of some to look at ideas and opinions that are different from everyone else's could lead to a better solution or idea. This does not mean that one side of an argument or the other is always right. Frequently, it is some combination of ideas that is the best solution. However, if you avoid any disagreement or dissent on project development and design ideas,

you will not have any useful discussions that will lead to a good product.

- Be willing to listen to ideas that are different from yours
- Be willing to listen to and consider opinions that aren't the same as those in the majority for a project group.
- Be willing to suggest ideas that are different. Despite the fear of what others may think of you.
- Be willing to take the time to discuss ideas and LET THEM EVOLVE with new ideas from all members of the group. The result may be a far stronger solution than any that could be created by one person in the group.
- Be willing to review things that everyone thought was decided. It is possible that something new has been discovered that will show everyone that the decision was wrong.
- And finally....if you are providing a dissenting opinion...be willing to accept that your opinion may not be the best for the group at this time if discussion does not move to new understandings.

B. Use Strengths Based Approach

When problems arise and the outlook for a project or a team looks bleak, try to avoid dwelling on all of the negative issues. Instead, look at the strengths the group or the project plan have and determine how those strengths can be used to handle the issues at hand. Inventory your resources and abilities to determine how you can turn the situation into a working solution.

A good example occurs in the *Princess Bride*.... The heroes wish to get into a castle guarded by sixty men. There are only three of them. Instead of despairing, they determine the assets available to them and attempt to find a solution to the problem. In this case, their assets include a wheelbarrow, a holocost cloak, a torch, one person's strength, one person's brain and the last person's ability to use the sword.

In a real world case, the project team was informed that management wanted a new set of requirements added to a project three weeks prior to the expected delivery of the final project. This project had been underway for 8 months, and such changes would certainly have been nearly impossible to implement in three weeks. Furthermore, there was considerable pressure on the group to succeed with the project (jobs were on the line). The group could have despaired and resigned their posts, going to new businesses and 'better jobs.' The group could have frantically tried to get the project to work with the new items. Or, the group, could band together and find an alternative.

In this example, the group had a number of strengths. First, the current project was very well documented and was on schedule for the delivery prior to the new demands. Second, the client group contact person had a good relationship with the client group who would be using the product. Using these strengths the group was able to ally themselves with the client group to present a counter proposal to management. This proposal utilized data from their documentation that showed time and resources necessary for parts of the project that had been completed that were similar to the new requirements. The group was able to show management that the request would require an additional five months of work. With the support of the client group, they proposed a solution that required delivery of the product as it was originally planned in three weeks. The new requirements would then become part of a new phase of the project that would undergo the normal processes for the following five months.

Team Development Exercise

Appendix II: Demographics Data Collection Tool

Demog. Coll.

Student ID _____
School BSU
Semester Spring

This questionnaire is part of a study being conducted to identify ways instruction in computer science can be improved. Your answers are confidential and will not effect your grade in any way. If you are uncomfortable with any question on this form, please leave it blank.

Gender: M/F **Year in School:** Fr So Jr Sr Grd **Age:**

Race (e.g. Native American, Caucasian, Hispanic, etc):

Of the following, **circle all responsibilities that apply** to you:

Part-time Employment Full-time Employment Parent/Caretaker
Internship Commute

Do you have **any learning disabilities** that you are aware of (e.g. dyslexia, reading, etc)? Please identify.

Do you have **any physical disabilities** that may impact your learning (e.g. eyesight, motor-skills, etc)? Please identify.

Your **average grade in high school** (please circle one):

A A/B B B/C C C/D D

Your **average grade in college** (please circle one):

A A/B B B/C C C/D D

Did you take **CS 1309** (Intro to Computer Science)? If so, **what semester?**

Fall '00 Spring '99 Other None

Your **grade in CS 1309** (Intro to Computer Science):

A B C D F Not Applicable

Do you plan on being a Computer Science (CS or CIS) **major?** (rate from 1 to 5)

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Computer Science (CS or CIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Management Information Systems (MIS) **major**?

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Management Information systems (MIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

Rate your level of comfort, from 1 to 5, with **computers**:

Difficult to work with 1 2 3 4 5 Easy to work with

Rate your level of comfort, from 1 to 5, with **math**:

Difficult to do 1 2 3 4 5 Easy to do

Rate your level of comfort, from 1 to 5, with **problem solving**:

Difficult to do 1 2 3 4 5 Easy to do

If you have programmed before, rate your level of comfort, from 1 to 5, with **programming**
(don't answer if you've never programmed):

Difficult to do 1 2 3 4 5 Easy to do

How useful do you think CS 1309 will be for your success in this class? (skip if no CS1309)

Wasted Time 1 2 3 4 5 Extremely Helpful

What were the **most useful** and **least useful parts** of the CS 1309 course to you?

What are you **most looking forward** to in this course?

What are you **most worried about** in this course?

Thank you for completing this survey. Your help is appreciated.

Appendix III: Exit Survey Data Collection Tool

Exit Survey

Student ID _____
School BSU
Semester Spring

This questionnaire is part of a study being conducted to identify ways instruction in computer science can be improved. Your answers are confidential and will not effect your grade in any way. If you are uncomfortable with any question on this form, please leave it blank.

After completing this course, do you plan on being a Computer Science (CS or CIS) **major**? (rate from 1 to 5)

No Maybe Not I Don't Know Maybe Yes Yes

After completing this course, do you plan on being a Computer Science (CS or CIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

After completing this course, do you plan on being a Management Information Systems (MIS) **major**?

No Maybe Not I Don't Know Maybe Yes Yes

After completing this course, do you plan on being a Management Information systems (MIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

Rate your level of comfort, from 1 to 5, with **computers**:

Difficult to work with 1 2 3 4 5 Easy to work with

Rate your level of comfort, from 1 to 5, with **math**:

Difficult to do 1 2 3 4 5 Easy to do

Rate your level of comfort, from 1 to 5, with **problem solving**:

Difficult to do 1 2 3 4 5 Easy to do

Rate your level of comfort, from 1 to 5, with **programming**:

Difficult to do 1 2 3 4 5 Easy to do

The following questions ask you to think about how a previous course (COMS 1309 – Intro to CS) has impacted your learning in this course.

How useful do you think COMS 1309 WAS for your success in this class? (skip if no COMS1309)

Wasted Time 1 2 3 4 5 Extremely Helpful

In your opinion, what were the **most useful parts** of the COMS 1309 course to you?

In your opinion, what were the **least useful parts** of the COMS 1309 course?

Is there anything that you think **should have been covered** in the COMS 1309 course that would have helped you in this programming course?

Do you have any suggestions that can help us to improve the COMS 1309 or this programming course (content, materials, order of topics, additional topics, methods of instruction that helped most, etc)?

Your help in completing this study has been greatly appreciated. Thank you very much for your time and willingness to share your opinions and to complete quizzes.

Appendix IV: Pretest Data Collection Tool

pre. A

Student ID _____
School BSU Class Section _____
Semester Fall
Instructor _____

These questions are part of a study being conducted to identify ways instruction in computer science can be improved. Please do your best to answer each question. **Show all work and sketches you use to attempt to solve the problem. It is important to this study to see how you attempt to solve the problems.** If you use scratch paper, please include it with this item (put your student ID on it).

1. You are given three mystery numbers, A, B and C. The only way you can learn about these numbers is by comparing two of them. When you compare, you are told which number of the two is bigger or if the two numbers are equal. Provide a method that will allow us to always put the three numbers in order – only using the knowledge we get from the results of ‘*compare*.’

2. Provide a step-by-step solution that will convert a time period given in minutes to a time period given in a combination of years, days, hours and minutes.

Appendix V: Posttest Data Collection Tool

post

Student ID _____
School BSU
Semester Spring

These questions are part of a study being conducted to identify ways instruction in computer science can be improved. Please do your best to answer each question. If you use scratch paper, please include it with this item (put your student ID on it).

1. Write a function that will swap the contents of two integer vectors. (when it is done, vector A will hold B's integers and B will hold A's).

2. Write a function (or functions) that receives a vector of integers and checks to see if the vector holds a palindrome. (A palindrome means that, when reversed, the array holds the same contents as it had before reversal. Example : 4,2,6,2,4 is still 4,2,6,2,4 when it is reversed.) The function will return true when it IS a palindrome and false when it is not.

3. Write a function to display a big X on the screen. The X will be a certain number of characters wide. The function will accept a single parameter (size of the X). Assume the size of the X will be between 3 and 10. A X of size 6 looks like :

X X (4 spaces between X's) And size 5 looks like X X (3 spaces)

```

X   X
 X X
  XX
 X X
X   X

```

```

 X X
  X
 X X
 X  X

```

4. You have been asked to write a new program that will help the school schedule their classes so that each class has a time, room and instructor and avoids conflicts. Write the appropriate function header definitions and declare major data variables/structures that you would use to accomplish this task.

Appendix VI: Informed Consent

CONSENT FORM FOR EDUCATIONAL RESEARCH

Effectiveness of Introductory Computer Science Curriculum

You are asked to participate in a research study conducted by Rob Faux, Ph.D. Candidate and the Computer Science department at Minnesota State University - Bemidji (BSU). This research is being conducted as a part of Mr. Faux's doctoral research and will be used as a tool to improve the curriculum offered by BSU. You were selected as a possible participant in this study because you are a participant in this course of study.

PURPOSE OF THE STUDY

The purpose of this research project is to evaluate the effectiveness of the Introduction to Computer Science (CS 1309) curriculum. Various approaches with respect to the content of the curriculum will be utilized in order to determine which combination of material works best to support learning. The BSU Computer Science department is interested in providing learners with the highest quality learning experience possible. This study will support this goal by working to identify approaches that prepare students for future courses in the Computer Science.

PROCEDURES

If you volunteer to participate in this study, I would ask you to do the following things:

- ◆ Share your opinions with respect to this course by completing evaluations at the beginning and end of this course.
- ◆ Participate in the completion of short assessment tools at the beginning and end of the course.
- ◆ Allow test and exercise results to be used as measurements of learning that will provide some feedback on curriculum effectiveness.

POTENTIAL RISKS AND DISCOMFORTS

There are no physical or emotional risks or discomforts anticipated with this research project beyond the normal effort exerted in a classroom environment. Any uses of test scores or exercise results will be coded so that individual learners are not identified by the researcher.

POTENTIAL BENEFITS TO SUBJECTS AND/OR TO SOCIETY

No immediate, direct benefits will be given to you for participation in this research study. However, the findings of this study may be beneficial to you in later courses, or to those who follow you as they take this sequence of courses at BSU. Further, findings of these studies may encourage other organizations to adopt more effective methods of teaching and learning in this area.

CONFIDENTIALITY

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. Your work will be used to determine your success in this class, as it would in most classroom situations. Confidentiality will be maintained by means of coding of your work so that the researcher is unaware of names and other identifying information. The information gained from this research project may be published in professional journals, shared in presentations, symposia, educational seminars and sessions without personal identifications of the participants.

PARTICIPATION AND WITHDRAWAL

You can choose whether or not to be a part of this study. If you volunteer to be in this study, you may withdraw at any time without consequences of any kind. You are not waiving any legal claims, rights or remedies because of your participation in this research study. You are reminded that participation in this study will not change your requirements for effort in this course of study.

IDENTIFICATION OF RESEARCHER

If you have any questions or concerns about the research, please feel free to contact:

Rob Faux, Ph.D. Candidate
2506 Townline Road
Decorah, IA 52101
rfaux@oneota.net

I understand the procedures and information described above. My questions have been answered to my satisfaction, and I agree to participate in this study. I have been given a copy of this form.

Signature of Participant

Date

Signature of Researcher

Date

Appendix VII: Instructor Information

Introduction to Computer Science Study

Demographics Survey

Thank you for being willing to participate in this study. It is our hope that some link to curriculum changes in COMS 100/CS 1309 Intro to Computer Science will be found regarding learning in the first programming course.

This study is serving as the dissertation in Computer Science Education for Rob Faux. Please feel to contact him at any point with concerns or questions.

- Do not distribute the questionnaire prior to completion of Informed Consent.
- You may do this before or after the Pre Test
- Ask students to only mark their student ID, not their names, on the questionnaire
- If a student expresses concern regarding any of the questions, you may remind them that they may skip a question
- Clarify questions for individual students as necessary.
- Remind students that there are questions on the front and back of the form.
- Collect all resulting papers and submit them to Marty Wolf (Bemidji) or Lee Cornell (Mankato). They will work with me to forward the items to my care.
- Data results of the pretest will be provided to Lee and Marty and will be available on request.

[Demographics \(Mankato\)](#)

[Demographics \(Bemidji\)](#)

[Return to Main Page](#)

[rfaux 8/29/00](#)

Introduction to Computer Science Study

Informed Consent

Thank you for being willing to participate in this study. It is our hope that some link to curriculum changes in COMS 100/CS 1309 Intro to Computer Science will be found regarding learning in the first programming course.

This study is serving as the dissertation in Computer Science Education for Rob Faux. Please feel to contact him at any point with concerns or questions.

- Please complete the Informed Consent before the Pre Test or Demographics Survey.
- Any student may opt to not participate in this study. If I do not have an informed consent form on file for a given Student ID, any information received regarding that person will be ignored.
- ***Please make sure students include their Student ID next to their signature.***
- Collect all resulting papers and submit them to Marty Wolf (Bemidji) or Lee Cornell (Mankato). They will work with me to forward the items to my care.

[Informed Consent \(Mankato\)](#)

[Informed Consent \(Bemidji\)](#)

[Return to Main Page](#)

[rfaux](#) 8/29/00

Introduction to Computer Science Study

Exit Survey

Thank you for being willing to participate in this study. It is our hope that some link to curriculum changes in COMS 100/CS 1309 Intro to Computer Science will be found regarding learning in the first programming course.

This study is serving as the dissertation in Computer Science Education for Rob Faux. Please feel to contact him at any point with concerns or questions.

- Please implement this survey at the end of the course. It may occur before or after the final. It may also occur before or after the post test.
 - Ask students to only mark their student ID, not their names, on the questionnaire
 - If a student expresses concern regarding any of the questions, you may remind them that they may skip a question
 - Clarify questions for individual students as necessary.
 - Remind students that there are questions on the front and back of the form.
 - Collect all resulting papers and submit them to Lee Cornell (Mankato) or Marty Wolf (Bemidji). They will work with me to forward the items to my care.
 - Data results of the pretest will be provided to Lee and Marty and will be available on request.
-

[Exit Survey \(Mankato\)](#)

[Exit Survey \(Bemidji\)](#)

[Return to Main Page](#)

[rfaux](#) 8/29/00

Introduction to Computer Science Study

Implementing Post Test

Thank you for being willing to participate in this study. It is our hope that some link to curriculum changes in COMS 100/CS 1309 Intro to Computer Science will be found regarding learning in the first programming course.

This study is serving as the dissertation in Computer Science Education for Rob Faux. Please feel to contact him at any point with concerns or questions.

You have two options for implementing this part of the study.

OPTION 1: Offer posttest as an event that will not be graded

*- All
Agreed
NOT TO DO THIS*

OPTION 2: Offer posttest as a part of the course final

- Distribute copies of post test A and B evenly throughout the classroom
- Allow learners 45 minutes to complete the test
- Ask students to only mark their student ID, not their names, on the post test
- Allow students to use scratch paper and ask them to put their student ID on these and include them with the posttest.
- There are (TBA) questions on this pretest. Please note this to students.
- Encourage students to break time down so that they have a chance to do some work on each problem
- Collect all resulting papers and submit them to Lee Cornell (Mankato) or Marty Wolf (Bemidji). They will work with me to forward the items to my care.
- Data results of the pretest will be provided to Lee and Marty and will be available on request.
- For curious students - answers to these questions will not be made available until after the use of these questions during the Spring Semester. (Summer of 2001)

[Post Test A](#)

[Post Test B](#)

[Return to Main Page](#)

rfaux 8/29/00

Introduction to Computer Science Study

Description

Research Question:

Will the integration of a set of beginning software engineering and problem solving concepts and techniques into a pre-programming computer science course impact the overall learning of computer science concepts and programming techniques for college and university learners?

Summary:

This study attempts to measure learning in the first programming course as it is influenced by the content of the preprogramming course. The researcher has provided base materials for new topics to be integrated into the existing preprogramming curriculum (which is currently based on Gersting & Schneider). For the next two semesters (Fall, 2000 and Spring, 2001), data will be collected in the first programming courses regarding learning. The majority of learners in the fall courses will not have been exposed to these new topics in the class environment. Most of the spring course attendees will have had this exposure. Various data collection tools are provided to allow the researcher to perform paired analysis.

Expectations of Preprogramming Course Faculty (COMS 100/CS 1309):

Faculty for the preprogramming courses are asked to integrate the topics and material provided by the researcher into this course. The materials provided by the researcher are provided to be base materials in a similar vein to the Gersting/Schneider text. Instructors are asked to incorporate these topics and provide them with appropriate weight and emphasis in the course during the Fall, 2000 semester. In other words, the instructor is responsible for selecting exercises, exam questions and class materials for these topics, using the materials provided by the researcher as the framework. Instructors may opt to continue to use these topics and materials in future semesters at their discretion.

Expectations of First Programming Course Faculty:

Faculty in the first preprogramming course are asked to implement several data collection devices during both the Fall, 2000 and Spring, 2001 terms. These tools include:

1. Informed Consent
2. Demographics Survey
3. Pretest
4. Mid-Term Exercise
5. Posttest
6. Exit Survey

Instructors are asked to provide class time for these events. Data analysis results will be provided to

participants on completion of the analysis. Each of these items will include only the student ID and school as identifying factors to allow paired analysis.

The Mid-Term Exercise and Posttest may be utilized as graded events, at the discretion of the faculty person. However, the grade will not be considered or viewed by the researcher. Similarly, the research analysis will not be available for grading purposes. If these two items are utilized as items for grading, the researcher requests that an unmarked COPY of the student's work be sent to him for analysis.

Contacts for this study at the respective schools are Marty Wolf (Bemidji) and Lee Cornell (Mankato). Please submit all materials to them in order to streamline forwarding to the researcher.

Please review guidance material for each data collection item prior to implementing them. If you have any concerns, questions or notice difficulties with the materials, please contact the researcher.

Researcher Biographical Information:

Rob Faux is currently a doctoral candidate with Union Institute, based in Cincinnati, Ohio. Rob's field of study is Computer Science Education, which will supplement a M.S. in Computer Science from MSU -Mankato and a B.A. in Computer Science and Mathematics from Luther College in Decorah, IA. Rob also serves as an Associate Faculty member for Open University in the field of Computing.

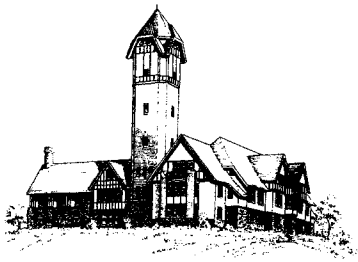
If you would like to know more about Rob's doctoral program or would like a copy of the dissertation/research proposal, please feel free to email him.

[Full Proposal](#)

[Return to Main Page](#)

rfaux 8/29/00

Appendix VIII: UIU Institutional Review Board Materials



The Union Institute

440 E. McMillan Street
Cincinnati, Ohio 45206-1925
513/861-6400 ♦ 800/486-3116
TDD 800/486-9968 ♦ FAX 513/861-0779

January 11, 2001

Mr. Robert Faux
2506 Town Line Road
Decorah, IA 52101

Re: IRB #TUI-0001
Impact of Preprogramming Course Curriculum Content on Student Learning in the First
Programming Course

Dear Mr. Fox:

The Institutional Review Board of The Union Institute, at its meeting of January 10, 2001, has approved your proposed research project.

Since your project is already under way under an approval issued by the IRB of Minnesota State University, The Union Institute's IRB has determined that this approval shall be concurrent with that of Minnesota State University. That is, the Union Institute IRB approves your study for a period of twelve months, beginning September 1, 2000 through August 31, 2001.

The IRB reserves the right to review your study as part of its continuing review process. Continuing reviews are typically scheduled in advance, however, the IRB may choose, under certain conditions, to not announce a continuing review. Please notify the IRB Chair when you have concluded your study.


Should you wish to make any substantive changes in your study design, funding source(s), consent processes, or any other aspect of the study that may affect study participants, you are required to submit proposed changes to the IRB for review. Should you require an extension on this approval, you are required to submit a request for extension to the IRB for review. The IRB reference number at the top of this letter should be used in any correspondence regarding your study.

On behalf of the IRB, I wish you success with your study and a satisfactory conclusion to your doctoral program with The Union Institute.

Sincerely,

Linda C. Van Volkenburgh
Acting Chair, Institutional Review Board

cc. Richard Genardi, Dean
Ben Davis, Core Faculty Advisor

Relationship to The Union Institute:	
<input type="checkbox"/> Faculty	<input checked="" type="checkbox"/> Learner
<input type="checkbox"/> Staff	<input type="checkbox"/> Other
College or Department:	
<input type="checkbox"/> College of Undergraduate Studies	<input type="checkbox"/> Office for Social Responsibility
<input checked="" type="checkbox"/> Graduate College, Arts & Sciences*	<input type="checkbox"/> Other
<input type="checkbox"/> School of Professional Psychology	
*Includes HBCU, UTD, and other doctoral initiatives	
Researcher's Mailing Address:	
2506 TOWNLINE ROAD DECOAH, IA 52101	
Researcher's Telephone Number (include area code): (319) 387 0582	
Researcher's E-mail Address: rfaux@oneota.net	
Co-Researcher(s), including institutional affiliations:	
General academic discipline guiding the research protocol (e.g. Psychology, Education, Sociology, History, etc.) EDUCATION (Computer Science)	
Principal Researcher: Sign and Date	
 Signature	Date 12/6/00

Section II. Funding Information
Complete this section if you intend to seek external funding. If no external funding will be sought, proceed to Section III. All research proposals involving external funding must be reviewed by the IRB.
Funding Source
Project Title (if different from Section I):
Principal Researcher (if different from Section I):
Type of Application:
<input type="checkbox"/> Grant <input type="checkbox"/> Subcontract <input type="checkbox"/> Contract <input type="checkbox"/> Fellowship
Date of Submission to Funding Source:

Section III. Collaborative Research

Complete this section only if the research will be done in collaboration with another institution. Participating institutions share responsibility for safeguarding the rights and welfare of human subjects and for complying with all applicable regulations. All applications for approval of collaborative research must be submitted to the Institutional Review Board.

Will The Union Institute be the lead institution for this collaborative research?
 Yes No

If yes, proceed to section IV. If no, complete this section

Lead Institution: Minnesota State University - Mankato

Contact Person: Rick Roiger / Tony Filipowich Date of IRB Review at Lead Institution: Sep 14, 2000

Lead institution IRB Recommendation: Petition for exemption accepted, study approved. See attached.

Attach copies of all documentation pertaining to IRB review at the lead institution, including the complete application as reviewed by the IRB and a copy of the letter approving the research.

Section IV. Subject Information

If any subjects of the proposed research are part of a protected population (children, pregnant women, prisoners, disabled) or if the research involves more than minimal risk, the proposal must be submitted to the IRB.

Subjects (check all that apply)

- Fetus in utero/non-viable fetuses/abortuses
- Newborns/infants (less than 2 years old)
- Children (aged 2-12)
- Adolescents (aged 13-18)
- Adults (over age 18)
- Pregnant women
- Other "special" populations (e.g. prisoners, mentally ill/disabled). Describe: Possible as part of non-recruited population.

Other (check all that apply):

- Use of investigational drugs or devices
- Information to be collected requires special sensitivity (e.g. substance abuse, sexual behavior)

Number of subjects: approx 200 Approximate total time commitment for each subject: 2 hours

Will subjects be compensated? Yes No

Form of compensation:

Amount of compensation:

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

I. General Information

a. Project Title

Impact of Preprogramming Course Curriculum Materials on Student Learning
in the First Programming Course

b. Principle Researcher

Mr. Robert Faux
2506 Town Line Road
Decorah, IA 52101
319-387-0582
rfaux@oneota.net

c. Proposed Study Dates

September 1, 2000 and May 30, 2001

d. Location of Project

Minnesota State University – Mankato
Computer and Information Sciences Department
PO Box 6800
Mankato, MN 56002-6800
Preprogramming course - COMS 100 (fall – curriculum content only)
First programming course - COMS 102 (fall & spring – data collection only)

Minnesota State University – Bemidji
Mathematics and Computer Science Department
1500 Birchmont Drive
Bemidji, MN 56601
Preprogramming course – CS 1309 (fall – curriculum content only)
First programming course – CS 2321 (fall & spring – data collection only)

e. Source of Funding

None

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

II. General Purpose of and Potential Benefits of Research Project

This study focuses on the impact particular curriculum content might have on how learners in the preprogramming course are able to perform in the first programming course. Topics have been added or expanded in the preprogramming course to include material on Problem Solving, Testing Algorithms, Flow Charts, Team Development, Software Engineering, and Diagramming. It is hypothesized that these new topics will provide tools that will better prepare learners for success in the programming course.

III. Project Description, Methods and Procedures

This study attempts to measure learning in the first programming course as it is influenced by the content of the preprogramming course. The researcher has provided base materials for new or expanded topics to be integrated into the existing preprogramming curriculum (which is currently based on Gersting & Schneider). For the next two semesters (Fall, 2000 and Spring, 2001), data will be collected in the first programming courses regarding learning quality. The majority of learners in the fall courses will not have been exposed to these new topics in the class environment. Most of the spring course attendees will have had this exposure. Various data collection tools are provided to allow the researcher to perform paired analysis on course learning.

Faculty for the preprogramming courses are asked to integrate the topics and material provided by the researcher into this course. The materials provided by the researcher are base materials in a similar vein to the Gersting/Schneider text. Instructors are asked to incorporate these topics and provide them with appropriate weight and emphasis in the course during the Fall, 2000 semester. In other words, the instructor is responsible for selecting exercises, exam questions and class materials for these topics, using the materials provided by the researcher as the framework. Instructors may opt to continue to use these topics and materials in future semesters at their discretion.

Faculty in the first preprogramming course are asked to implement several data collection devices during both the Fall, 2000 and Spring, 2001 terms. These tools include: Demographics Survey, Pretest, Posttest, and Exit Survey. Instructors are asked to provide class time for these events. Data analysis results will be provided to participants on completion of the analysis. Each of these items will include only a student identifier to allow for paired analysis. No direct identification will be included that will allow the researcher to track actual results to the student.

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

IV. Description of Subjects, Recruitment and Compensation

Data collection will be performed on participants in the Fall, 2000 and Spring, 2001 semester courses offered by the Computer and Information Sciences Departments at Minnesota State University at Mankato and Bemidji. All participants are college-aged individuals and should be a representative population of MSU students. The sample size is estimated to be approximately 200 total students. Individuals with disabilities, who are members of other protected populations may be participants in this study, but are not recruited for these reasons.

Participant recruitment is based solely on course enrollment. Members of participating classes are not aware of the existence of the study at the point of enrollment. No effort is made to promote participation in this study.

Participants are not compensated for participation in this study. Participants are not required to perform additional tasks outside of classroom activities.

V. Protection of Subject's Rights/ Potential Risks and Confidentiality

Students will be informed of their rights with respect to the participation of this study via the Informed Consent form included with this document. The form includes a description of the intent and purpose of the study, delineation of potential risks and benefits, and their rights regarding participation. Students are given the right to remove themselves from the study at any point during data collection.

Consent documents will be maintained with the data collection materials with the researcher (contact person) listed on this form. The researcher will maintain data collection documents. Data analysis results will be available on request. Data that may identify an individual will be destroyed and not used for data analysis.

A student identifier marks student materials rather than a name for the purposes of paired analysis in this study. The researcher will not have access to the names or identities of those who are performing the work. Instructors in these courses will be provided with access to post analysis data, but specific analysis to a given individual's material will not be allowed.

There is no direct risk to participants in this study. Results of data collection will not be used to determine success in the course. Participants will not be singled out for participatory reasons and study content will be integrated into normal course workings.

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

VI. Signature

In making this application, I certify that I have read and understand the Policies and Procedures for Projects that Involve Human Subjects, and that I intend to comply with the letter and spirit of the Union Institute's policies. Significant changes in the protocol will be submitted to the IRB for written approval prior to these changes being put into practice. Informed consent/assent records of the participants will be kept for at least three (3) years after the completion of the research.

Principle Investigator

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

CONSENT FORM FOR EDUCATIONAL RESEARCH

Effectiveness of Introductory Computer Science Curriculum

You are asked to participate in a research study conducted by Rob Faux, Ph.D. Candidate and the Computer and Information Sciences department at Minnesota State University - Mankato (MSU). This research is being conducted as a part of Mr. Faux's doctoral research and will be used as a tool to improve the curriculum offered by MSU. You were selected as a possible participant in this study because you are a participant in this course of study.

PURPOSE OF THE STUDY

The purpose of this research project is to evaluate the effectiveness of the Introduction to Computer Science (COMS 100) curriculum. Various approaches with respect to the content of the curriculum will be utilized in order to determine which combination of material works best to support learning. The MSU Computer and Information Sciences department is interested in providing learners with the highest quality learning experience possible. This study will support this goal by working to identify approaches that prepare students for future courses in the Computer and Information Sciences.

PROCEDURES

If you volunteer to participate in this study, I would ask you to do the following things:

- ◆ Share your opinions with respect to this course by completing evaluations at the beginning and end of this course.
- ◆ Participate in the completion of short assessment tools at the beginning and end of the course.
- ◆ Allow test and exercise results to be used as measurements of learning that will provide some feedback on curriculum effectiveness.

POTENTIAL RISKS AND DISCOMFORTS

There are no physical or emotional risks or discomforts anticipated with this research project beyond the normal effort exerted in a classroom environment. Any uses of test scores or exercise results will be coded so that individual learners are not identified by the researcher.

**Impact of Preprogramming Course Curriculum Materials
on Student Learning in the First Programming Course**

POTENTIAL BENEFITS TO SUBJECTS AND/OR TO SOCIETY

No immediate, direct benefits will be given to you for participation in this research study. However, the findings of this study may be beneficial to you in later courses, or to those who follow you as they take this sequence of courses at MSU. Further, findings of these studies may encourage other organizations to adopt more effective methods of teaching and learning in this area.

CONFIDENTIALITY

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. Your work will be used to determine your success in this class, as it would in most classroom situations. Confidentiality will be maintained by means of coding of your work so that the researcher is unaware of names and other identifying information. The information gained from this research project may be published in professional journals, shared in presentations, symposia, educational seminars and sessions without personal identifications of the participants.

PARTICIPATION AND WITHDRAWAL

You can choose whether or not to be a part of this study. If you volunteer to be in this study, you may withdraw at any time without consequences of any kind. You are not waiving any legal claims, rights or remedies because of your participation in this research study. You are reminded that participation in this study will not change your requirements for effort in this course of study.

IDENTIFICATION OF RESEARCHER

If you have any questions or concerns about the research, please feel free to contact:

Rob Faux, Ph.D. Candidate
2506 Townline Road
Decorah, IA 52101
rfaux@oneota.net

I understand the procedures and information described above. My questions have been answered to my satisfaction, and I agree to participate in this study. I have been given a copy of this form.

Signature of Participant (student ID)

Date

Signature of Researcher

Date

Appendix IX: MSU Institutional Review Board Materials



September 14, 2000

Robert Faux
2506 Town Line Road
Decorah, IA 52101

From: Anthony Filipovitch, IRB Administrator
MSU Institutional Review Board (IRB)

Re: IRB #1248
Impact of the Integration of Software Engineering Concepts and Problem Solving
Techniques in Preprogramming Course Curriculum Materials on Student Learning in the
First Programming Course

On behalf of the Institutional Review Board I wish you success with your study. Remember that you must seek approval for any changes in your study, its design, funding source, consent process, or any part of the study that may affect participants in the study. Should any of the participants in your study suffer a research-related injury or other harmful outcome, you are required to report them to the IRB as soon as possible.

Approval of your study is for one calendar year from the approval date. When you complete your data collection, or should you discontinue your study, you must notify the IRB. Please include your log number with any correspondence with the IRB.

The IRB reserves the right to review each study as part of its continuing review process. Continuing reviews are usually scheduled, however under some conditions the IRB may choose not to announce a continuing review.

Sincerely,

A handwritten signature in black ink, appearing to read "Anthony J. Filipovitch".

Anthony J. Filipovitch
IRB Administrator

Cc: file

INSTITUTIONAL REVIEW BOARD
COLLEGE OF GRADUATE STUDIES AND RESEARCH
125 WIGLEY ADMINISTRATION CENTER · MANKATO, MN 56001
PHONE 507-389-6310 · FAX 507-389-5974

An affirmative action/equal opportunity university.

● Benjamin R.H. Davis ●

FAX TRANSMITTAL

Date September 12, 2000

of pages _2_

To: Tony Filipovitch

FAX # 507 - 389 - 5974

Re: IRB fasttrack - Rob Faux

● P.O. Box 27 ● Bolinas, California 94924 ●

Dr. Filipovitch,

I have included a copy of the committee sign-off sheet concerning Rob Faux's proposed research project. His doctoral committee has approved his study, including his statement and forms related to the protection of human subjects. In the TUI academic process, the doctoral committee is responsible for assuring the candidate fully informs the research respondents and protects their rights as participants.

Thank you for your assistance with Rob's research.

Ben Davis, Ph.D.
Professor
Graduate School of Interdisciplinary Arts and Sciences
The Union Institute
442 McMillan Ave
Cincinnati, OH 45201

Core Faculty, The Union Institute Graduate College
PHONE & FAX: 415/868/9313

Application for the Conduct of Research Involving Human Subjects Guidelines

I. General Information

a. Principle Investigators

Dr. Richard Roiger
Computer and Information Sciences Dept
PO Box 8400
Minnesota State University – Mankato
Mankato, MN 56002-8400
507-389-2968
roiger@krypton.mankato.msus.edu

b. Co-Investigator

Mr. Robert Faux
2506 Town Line Road
Decorah, IA 52101
319-387-0582
rfaux@oneota.net

c. Whom the IRB should contact regarding proposal

Robert Faux (as above)

d. Project Title

Impact of the Integration of Software Engineering Concepts and Problem Solving Techniques in Preprogramming Course Curriculum Materials on Student Learning in the First Programming Course

e. Proposed Study Dates

Fall, 2000 and Spring, 2001 terms

f. Location of Project

Minnesota State University – Mankato
Computer and Information Sciences Department
Preprogramming course - COMS 100 (fall – curriculum content only)
First programming course - COMS 102 (fall & spring – data collection only)

g. Source of Funding

None

II. General Purpose of Research Project

This study focuses on the impact particular curriculum content might have on how learners in the preprogramming (COMS 100) course are able to perform in the first programming course (COMS 102). Topics in COMS 100 have been added or expanded to include material on Problem Solving, Testing Algorithms, Flow Charts, Team Development, Software Engineering, and Diagramming. It is hypothesized that these new topics will provide tools that will better prepare learners for success in the programming course.

III. Project Description

This study attempts to measure learning in the first programming course as it is influenced by the content of the preprogramming course. The researcher has provided base materials for new or expanded topics to be integrated into the existing preprogramming curriculum (which is currently based on Gersting & Schneider). For the next two semesters (Fall, 2000 and Spring, 2001), data will be collected in the first programming courses regarding learning quality. The majority of learners in the fall courses will not have been exposed to these new topics in the class environment. Most of the spring course attendees will have had this exposure. Various data collection tools are provided to allow the researcher to perform paired analysis on course learning.

Faculty for the preprogramming courses are asked to integrate the topics and material provided by the researcher into this course (see appendix C). The materials provided by the researcher are base materials in a similar vein to the Gersting/Schneider text. Instructors are asked to incorporate these topics and provide them with appropriate weight and emphasis in the course during the Fall, 2000 semester. In other words, the instructor is responsible for selecting exercises, exam questions and class materials for these topics, using the materials provided by the researcher as the framework. Instructors may opt to continue to use these topics and materials in future semesters at their discretion.

Faculty in the first preprogramming course are asked to implement several data collection devices during both the Fall, 2000 and Spring, 2001 terms. These tools include: Demographics Survey, Pretest, Mid-Term Exercise, Posttest (see Appendix B). Instructors are asked to provide class time for these events. Data analysis results will be provided to participants on completion of the analysis. Each of these items will include only the student identifier as identifying factors to allow paired analysis.

IV. Description of Subjects

Data collection will be performed on participants in the Fall, 2000 and Spring, 2001 semester COMS 102 courses offered by the Computer and Information Sciences Department at Minnesota State University at Mankato. All participants are college-aged individuals and should be a representative population of MSU students. The sample size of this population is based on roughly 25 students per course section of COMS 102. Fall semester will include a rough sample of 150 students.

V. Protection of Subject's Rights

Students will be informed of their rights with respect to the participation of this study via the Informed Consent form included as Appendix A in this document. The form includes a description of the intent and purpose of the study, delineation of potential risks and benefits, and their rights regarding participation. Students are given the right to remove themselves from the study at any point during data collection.

Consent documents will be maintained with the data collection materials with the researcher (contact person) listed on this form. Principle investigators will have access to these documents on request.

A student identifier marks student materials rather than a name for the purposes of paired analysis in this study. The researcher will not have access to the names or identities of those who are performing the work. Instructors in these courses will be provided with access to post analysis data, but specific analysis to a given individual's material will not be allowed.

VI. Signatures

In making this application, I certify that I have read and understand the Policies and Procedures for Projects that Involve Human Subjects, and that I intend to comply with the letter and spirit of the University Policy. Significant changes in the protocol will be submitted to the IRB for written approval prior to these changes being put into practice. Informed consent/assent records of the participants will be kept for at least three (3) years after the completion of the research.

Principle Investigators

Co-Investigator

Appendix A: Informed Consent

The following will be presented to each participant in the Fall, 2000 and Spring 2001 COMS 102 courses. Participants who do not opt to sign this form will not be included in data analysis.

CONSENT FORM FOR EDUCATIONAL RESEARCH

Effectiveness of Introductory Computer Science Curriculum

You are asked to participate in a research study conducted by Rob Faux, Ph.D. Candidate and the Computer and Information Sciences department at Minnesota State University - Mankato (MSU). This research is being conducted as a part of Mr. Faux's doctoral research and will be used as a tool to improve the curriculum offered by MSU. You were selected as a possible participant in this study because you are a participant in this course of study.

PURPOSE OF THE STUDY

The purpose of this research project is to evaluate the effectiveness of the Introduction to Computer Science (COMS 100) curriculum. Various approaches with respect to the content of the curriculum will be utilized in order to determine which combination of material works best to support learning. The MSU Computer and Information Sciences department is interested in providing learners with the highest quality learning experience possible. This study will support this goal by working to identify approaches that prepare students for future courses in the Computer and Information Sciences.

PROCEDURES

If you volunteer to participate in this study, I would ask you to do the following things:

- ◆ Share your opinions with respect to this course by completing evaluations at the beginning and end of this course.
- ◆ Participate in the completion of short assessment tools at the beginning and end of the course.
- ◆ Allow test and exercise results to be used as measurements of learning that will provide some feedback on curriculum effectiveness.

POTENTIAL RISKS AND DISCOMFORTS

There are no physical or emotional risks or discomforts anticipated with this research project beyond the normal effort exerted in a classroom environment. Any uses of test scores or exercise results will be coded so that individual learners are not identified by the researcher.

POTENTIAL BENEFITS TO SUBJECTS AND/OR TO SOCIETY

No immediate, direct benefits will be given to you for participation in this research study. However, the findings of this study may be beneficial to you in later courses, or to those who follow you as they take this sequence of courses at MSU. Further, findings of these studies may encourage other organizations to adopt more effective methods of teaching and learning in this area.

CONFIDENTIALITY

Any information that is obtained in connection with this study and that can be identified with you will remain confidential and will be disclosed only with your permission or as required by law. Your work will be used to determine your success in this class, as it would in most classroom situations. Confidentiality will be maintained by means of coding of your work so that the researcher is unaware of names and other identifying information. The information gained from this research project may be published in professional journals, shared in presentations, symposia, educational seminars and sessions without personal identifications of the participants.

PARTICIPATION AND WITHDRAWAL

You can choose whether or not to be a part of this study. If you volunteer to be in this study, you may withdraw at any time without consequences of any kind. You are not waiving any legal claims, rights or remedies because of your participation in this research study. You are reminded that participation in this study will not change your requirements for effort in this course of study.

IDENTIFICATION OF RESEARCHER

If you have any questions or concerns about the research, please feel free to contact:

Rob Faux, Ph.D. Candidate
2506 Townline Road
Decorah, IA 52101
rfaux@oneota.net

I understand the procedures and information described above. My questions have been answered to my satisfaction, and I agree to participate in this study. I have been given a copy of this form.

Signature of Participant (student ID)

Date

Signature of Researcher

Date

Appendix B: Data Collection Tools for COMS 102

Included here are the Demographics Survey and a condensed pretest (space for answers removed). Additional material may be viewed at <http://www.oneota.net/~rfaux/introcs/instructor/main.htm>

Materials not in this document are either exercises or test questions that may be found in any first programming course. They have been selected as questions for which content analysis will be made on learning. Please visit the web site if you wish to view these materials.

Demog. Coll.

Student Identifier _____
School MSU Class Section _____
Semester Fall
Instructor _____

This questionnaire is part of a study being conducted to identify ways instruction in computer science can be improved. Your answers are confidential and will not effect your grade in any way. If you are uncomfortable with any question on this form, please leave it blank.

Gender: M/F Year in School: Fr So Jr Sr Grd Age:

Race (e.g. Native American, Caucasian, Hispanic, etc):

Of the following, circle all responsibilities that apply to you:

Part-time Employment Full-time Employment Parent/Caretaker
Internship Commute

Do you have any learning disabilities that you are aware of (e.g. dyslexia, reading, etc)? Please identify.

Do you have any physical disabilities that may impact your learning (e.g. eyesight, motor-skills, etc)? Please identify.

Your average grade in high school (please circle one):

A A/B B B/C C C/D D

Your average grade in college (please circle one):

A A/B B B/C C C/D D

Your grade in COMS 100 (Intro to Computer Science):

A B C D F Not Applicable

Do you plan on being a Computer Science (CS or CIS) major? (rate from 1 to 5)

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Computer Science (CS or CIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Management Information Systems (MIS) **major**?

No Maybe Not I Don't Know Maybe Yes Yes

Do you plan on being a Management Information systems (MIS) **minor**?

No Maybe Not I Don't Know Maybe Yes Yes

Rate your level of comfort, from 1 to 5, with **computers**:

Difficult to work with 1 2 3 4 5 Easy to work with

Rate your level of comfort, from 1 to 5, with **math**:

Difficult to do 1 2 3 4 5 Easy to do

Rate your level of comfort, from 1 to 5, with **problem solving**:

Difficult to do 1 2 3 4 5 Easy to do

If you have programmed before, rate your level of comfort, from 1 to 5, with **programming**
(don't answer if you've never programmed):

Difficult to do 1 2 3 4 5 Easy to do

How useful do you think COMS 100 will be for your success in this class? (skip if no COMS100)

Wasted Time 1 2 3 4 5 Extremely Helpful

What were the **most useful** and **least useful parts** of the COMS 100 course to you?

What are you **most looking forward** to in this course?

What are you **most worried about** in this course?

pre. A

Student ID _____
School MSU Class Section _____
Semester Fall
Instructor _____

These questions are part of a study being conducted to identify ways instruction in computer science can be improved. Please do your best to answer each question. Show all work and sketches you use to attempt to solve the problem. It is important to this study to see how you attempt to solve the problems. If you use scratch paper, please include it with this item (put your student ID on it).

1. You are given three mystery numbers, A, B and C. The only way you can learn about these numbers is by comparing two of them. When you compare, you are told which number of the two is bigger or if the two numbers are equal. Provide a method that will allow us to always put the three numbers in order – only using the knowledge we get from the results of ‘compare.’
2. Provide a step by step solution that will convert a time period given in minutes to a time period given in a combination of years, days, hours and minutes.
3. You have been given the task of programming a game of BINGO (or GO, or CHECKERS, or CHESS – pick a game you KNOW) into a computer. Take a few minutes and describe your strategy for attempting to program this problem. You do not need to do any programming, just explain how you will organize your solution.
4. A traveler undertakes to walk, alone and without help, across a desert entirely lacking in resources. Every 20 kilometers on the 100-kilometer trail there is a shelter. 20 kilometers is exactly what the traveler can walk in 1 day. The traveler can only carry 3 days’ food at a time. He can stock food only at the shelters and can get food only at the beginning or end of the desert trail. How many days does it take him to cross the desert?

Appendix X: BSU Institutional Review Board Materials

Date: Wed, 30 Aug 2000 16:51:42 -0500 (CDT)
From: "Marty J. Wolf" <mjwolf@whitetail.bemidji.msus.edu>
To: Rob Faux <rfaux@oneota.net>
cc: Lee Cornell <cornell@krypton.mankato.msus.edu>
Subject: Re: FW: IRB Coordinator at BSU

Rob,

I have checked and the person on our campus to check with is Joan. I don't even know her last name. Her number is 218 755 3732. She has all of the forms that need to be filled out. She claims they are quite long. :^(

You may want to try to Dean David Larkin (I think Joan in his secretary), before proceeding with her. He can be reached at that number, too. I may see him tonite and if I do, I will try to get him up to speed on your project.

Regards,

MJ

--

*Approval of
receipt of
copies of
msu materials.*

Marty J. Wolf mjwolf@acm.org
Math & CS Department mjwolf@whitetail.bemidji.msus.edu
Bemidji State University Office: (218)755-2825
1500 Birchmont Drive, Box 23 Fax: (218)755-2822
Bemidji MN 56601 <http://whitetail.bemidji.msus.edu/mjwolf>

*\$20
incentive
only
\$0.150/month*

Appendix XI: BSU Data

GENERAL DATA ANALYSIS SHEET

CONTROL GROUP FALL 2000

COMPLETED COURSE & HAD CS 0

GROUP C1

ID	excomp	exmath	exprob	exprog	excs1309	preq1scr	preq1sty	preq1met	preq2scr	preq2sty	preq2met	preq3scr	preq3sty	preq3met	preq4scr	preq4sty	preq4met	post1	p1note	post2	p2note	post3	p3note	post4	p4note	pretot1	pretot2	posttot
c101	5	4	4	4	4	3	2	psu	3	3	mat	2	2	wrd	2	1	wrd/pic	4	2	notcode	2	notcode	2	2	pic	9	8	12
c102	4	4	4	4	4	1	2	3	wrd	4	3	wrd	2	3	wrd	2	0	none	2	1	notcode	2	notcode	2	1	9	9	7
c103	5	3	3	3	3	3	3	3	wrd	4	3	psu/wrd	4	3	2	hpic	4	0	1	1	1	1	0	10	8	6	6	
c104	4	4	4	4	4	4	5	4	wrd	3	3	wrd	1	2	wrd	4	3	pic	5	0	0	1	1	0	13	12	6	
c105	5	4	5	2	4	4	4	4	psu/wrd	4	3	mat/psu	4	3	2	hpic	0	2	notcode	0	0	1	notcode	11	9	3	3	
c106	4	3	3	3	4	4	4	5	psu	4	4	psu	1	3	wrd	4	2	hpic	4	3	3	3	4	0	9	10	10	
c107	3	3	3	1	3	3	3	3	wrd	4	4	wrd	3	4	wrd	3	2	hpic	2	0	1	1	1	13	13	4	4	
c108	3	3	3	3	1	3	4	psu	3	3	psu	1	2	wrd	3	1	wrd	2	1	1	0	1	1	11	10	4	4	
c109	5	4	4	4	1	4	4	psu	3	3	psu/wrd	1	3	wrd	3	1	hpic	3	3	1	1	1	wrd	11	11	8	8	
c110	5	4	4	4	2	5	3	wrd	3	3	mat/wrd	2	2	wrd	1	0	none	5	1	0	0	1	0	11	8	6	6	
c111	3	5	4	4	3	3	4	psu	3	3	mat/psu	3	3	wrd	3	1	1	pic	5	4	psu	4	3	10	11	16	16	
c112	3	5	4	4	4	3	4	psu	5	4	psu	1	1	mat	5	1	1	mat	5	1	1	1	1	11	12	8	8	
c113	5	5	4	3	3	4	2	wrd	1	2	wrd	1	1	wrd	1	1	wrd	1	1	0	1	1	1	7	8	3	3	
c114	5	4	4	4	4	4	4	psu	3	4	psu	2	3	psu	2	3	psu	4	1	2	1	1	1	9	11	9	9	
c115	5	5	4	4	3	3	3	psu/wrd	2	3	psu/wrd	2	3	wrd	2	1	pic	5	1	3	3	3	3	8	10	14	14	
c116	4	4	4	5	4	5	4	3	dia/wrd	3	4	psu	2	3	wrd	2	3	wrd/pic	3	3	1	1	1	11	13	8	8	
c117	2	3	3	3	2	3	4	psu	3	3	mat	3	3	wrd/pic	3	3	3	wrd/pic	2	1	notcode	1	1	9	10	5	5	
c118	4	4	4	4	5	3	2	wrd	3	3	psu/wrd	2	2	psu/wrd	2	1	hpic	3	1	1	1	0	8	7	5	5		
c119	5	5	5	5	4	4	4	psu	3	3	mat/psu	2	1	hpic	5	4	4	3	3	1	0	1	9	8	13	13		
c120	3	3	3	3	4	5	4	psu	5	4	mat	1	1	1	1	1	1	1	0	1	1	1	10	8	5	5		
c121	5	5	5	5	1	1	2	pic/wrd	1	1	mat	1	1	wrd	4	5	wrd	2	2	1	0	1	7	9	4	4		
c122	5	4	4	5	3	3	4	psu	5	4	psu/wrd	1	1	wrd	1	0	none	1	2	2	2	2	10	9	5	5		
c123	5	4	4	4	4	3	4	psu	5	4	psu	1	1	wrd	1	3	wrd/pic	4	4	3	2	2	10	12	11	11		
c124	5	5	5	5	3	4	2	psu/wrd	4	4	psu	2	3	wrd	1	1	wrd/pic	4	2	2	3	3	11	10	9	9		
count	24	24	24	24	20	24	24	wrd	23	23	wrd	19	19	wrd	21	21	wrd	24	notcode	24	notcode	24	24	notcode	24	24	24	24
mean	4.25	4.04	3.96	3.71	2.95	3.48	3.38	6	3.35	3.22	4	1.63	2.37	16	2.19	1.62	3	3.25	1	1.66	3	1.42	0.92	1	9.88	9.75	7.46	7.46
stdev	0.94	0.75	0.69	0.95	1.28	0.93	0.88	psu	1.11	0.74	psu	0.68	0.90	psu	1.21	1.24	hpic	1.48	1.39	psu	1.06	0.88	wrd	1.57	1.87	3.81	3.81	
var	0.89	0.56	0.48	0.91	1.63	0.87	0.77	13	1.24	0.54	7	0.47	0.80	2	1.46	1.55	6	2.20	1.94	1	1.12	0.78	2	2.46	3.60	13.04	13.04	
skew	-0.89	-0.07	0.93	-0.99	-0.24	-0.40	-0.43	psu/wrd	3	-0.34	-1.13	psu/wrd	0.63	-0.34	psu/wrd	0.35	0.99	pic	3	-0.47	0.68	0.72	1.01	pic	0.08	0.05	0.87	0.87
5s	13	7	5	4	2	3	1	dia/wrd	4	0	mat/wrd	0	0	0	0	1	none	8	1	0	0	0	1	0	0	0	0	
4s	5	11	13	12	5	8	12	1	5	8	1	0	1	4	0	3	6	3	1	4	0	0	0	0	0	0	0	
3s	5	6	6	6	7	11	6	pic/wrd	11	13	mat/psu	2	9	5	4	wrd/pic	4	4	3	3	3	2	2	2	2	2	2	
2s	1	0	0	1	2	1	5	1	1	1	3	8	5	3	4	5	5	2	5	2	2	2	2	2	2	2	2	
1s	0	0	0	1	4	1	0	2	1	mat	9	4	9	9	mat	2	2	9	9	11	12	12	12	12	12	12	12	
COMPLETED but Did NOT have CS 0																												
n101	5	4	5	5	4	3	4	psu	3	4	mat/psu	4	4	wrd	4	3	wrd	5	1	1	1	1	4	14	15	11	11	
DID NOT COMPLETE, but DID have CS 0																												
c150								3	4	psu	5	4	psu	1	1	wrd	1	3	wrd/pic							10	12	12
c151								3	4	psu	2	3	mat/psu	2	1	wrd	1	0	none								8	8
c152								4	5	psu/prg	5	4	psu	4	2	1	hpic	4	4	4	4	4	4	11	10	11	10	
c153								3	3	psu/wrd	4	2	mat	3	4	psu	1	1	hpic								9	9
c154								4	4	psu	5	5	psu														5	5
c155								2	2	psu/wrd	2	2	mat/psu														5	5
c156								4	3	psu	3	4	psu														8	8
c157								2	3	psu	1	2	psu	1	2	wrd	1	0	none								6	7
count								8	8	psu/prg	8	8	5	5	wrd	7	7	7	7	7	7	7	7	7	7	8	8	8
mean								3.13	3.50	1	3.38	3.25	2.20	2.40	4	1.14	1.00									8.88	9.13	9.13
stdev								0.83	0.93	psu	1.60	1.16	psu	1.30	1.52	psu	0.38	1.00	hpic							3.27	2.85	2.85
var								0.70	0.86	5	2.55	1.36	5	1.70	2.30	1	0.14	1.00	3							10.70	6.13	6.13
skew								-0.28	0.00	psu/wrd	-0.26	0.09		0.54	0.32		2.65	1.40								0.67	0.47	0.47
5s								0	1	2	3	1	0	0	0	0	0	none										
4s								3	3	1	3	mat/psu	1	2	1	2	0	0	2									
3s								3	3	1	1	1	2	1	0	0	0	1	wrd/pic									
2s								2	1	2	3	mat	1	1	1	0	1	0	2									
1s								0	0	1	0	1	2	1	2	1	4	4										

2.27 1.68
NS vs
1.14 1.00

2.61
NS vs
1.62 2.25

ID	excomput	exmath	exprobsol	exprop	excs1309	preq1scr	preq1sty	preq1mat	preq2scr	preq2sty	preq2met	preq3scr	preq3sty	preq3met	preq4scr	preq4sty	preq4met	post1	p1note	post2	p2note	post3	p3note	post4	p4note	pretot1	pretot2	posttot	
I11	5	4	4	4	4	3	4	psu	2	4	psu	3	3	wrd	4	4	pic/wrd	5	pic	2	wrd		4	pic	0		12	15	11
I12	4	3	3	2	4	3	4	psu	3	2	mat	2	2	pic/wrd	1	2	wrd	3		2			2	pic	1		9	10	8
I13	5	5	4	4	5	1	4	psu	4	4	psu	1	2	wrd	4	3	wrd	1		1		1		4	wrd	10	13	7	
I14	5	3	3	5	4	3	4	psu	4	3	psu	4	3	wrd	4	3	pic/wrd	5		3		3		2	wrd	15	13	13	
I15	3	5	2	1	2	1	2	wrd	1	1	psu/wrd	1	1	wrd	1	0	none	5		2		2		3		4	4	12	
I16	5	4	5	5	5	4	3	psu	4	4	psu	2	2	wrd	0	1	wrd	4		4		1		3		10	10	12	
I17	4	4	5	3	5	5	4	psu	2	3	psu	2	2	wrd	3	4	pic	5		5		1		2		12	13	13	
I18	4	4	4	3	5	2	4	psu	4	4	psu	1	2	wrd	4	3	mat/wrd	3		1		1		1		11	13	6	
I19	5	5	4	3	3	3	4	psu	3	2	mat/psu	1	1	wrd	2	3	pic	1		2		2		2		9	10	7	
I20	5	5	5	4	4	5	4	psu	2	3	mat/wrd	2	2	pic/wrd	5	5	pic	5		4		2	pic	1		14	14	12	
I21	5	3	4	5	1	3	4	pic/psu	4	3	mat/psu	3	4	wrd	4	5	pic/wrd	5		5		0		2		14	16	12	
I22	3	3	4	4	5	4	4	psu	2	2	wrd	1	1	wrd	3	3	mat/psu	1		3		1		1	pic	7	10	5	
I23	5	4	4	4	5	3	3	psu	2	2	wrd	1	1	psu/wrd	2	1	wrd	1		2		4		0		5	5	1	
I24	5	4	4	4	4	3	3	psu	3	4	psu	1	1	wrd	1	2	wrd	5		4		4		0		8	10	16	
I25	5	5	4	4	4	1	3	psu/wrd	4	2	psu/wrd	1	2	wrd	1	0	none	5		2		2	pic	3		7	7	9	
count	15	15	15	15	15	14	14	wrd	15	15	wrd	15	15	wrd	14	14	wrd	15		15	wrd	15		15	wrd	15	15	15	
mean	4.53	4.07	3.93	3.67	4.09	2.93	3.64	1	2.93	2.93	1	1.73	2.13	12	2.57	2.57	5	3.73		2.53	1	1.73		1.60	2	9.80	10.87	9.60	
stdev	0.74	0.80	0.80	1.11	1.20	1.33	0.63	psu	1.03	0.96	psu	0.96	0.83	pic/wrd	1.60	1.65	pic	1.62	pic	1.55		1.22	pic	1.30		3.28	3.50	3.91	
var	0.55	0.64	0.64	1.24	1.43	1.76	0.40	11	1.07	0.92	7	0.92	0.70	2	2.57	2.73	3	2.64	1	2.41		1.50	6	1.69		10.74	12.27	15.26	
skew	-1.33	-0.13	-0.84	-1.02	-1.45	-0.08	-1.69	psu/wrd	-0.30	-0.41	psu/wrd	1.17	0.58	psu/wrd	-0.09	-0.15	pic/wrd	-0.89		0.26		0.56		0.20		-0.10	-0.60	-0.58	
5s	10	5	3	3	6	2	0	pic/psu	0	0	mat/wrd	0	0	1	2	mat/wrd	8		2		0		0						
4s	3	6	9	7	6	2	10	psu	1	6	1	1	1	1	5	2	1	1	1	3		2		1					
3s	2	4	2	3	1	6	3	5	mat/psu	2	3	3	1	4	none	3	1	1	3		1		3						
2s	0	0	1	1	1	1	1	5	4	3	4	8	3	2	2	0	5	5	4	5		5	4	4					
1s	0	0	0	1	1	3	0	1	mat	8	3	4	2	mat	3	3	3	3	3	3		5	3	3					
COMPLETED and did not have treatment CS 0																													
tc1	4	3	3	3	4	2	3	psu	2	3	psu	1	3	wrd	4	4	pic/wrd	4		2		3		0		9	13	9	
tc2	5	3	4	3	3	3	3	3	psu	4	4	3	wrd	2	3	wrd	2	1	hpic	3		3		0	2		11	11	8
tc3	3	4	4	2	2	3	3	wrd	4	3	wrd	5	3	3	wrd	4	3	wrd	5		3		1	2		16	12	11	
count	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
mean	4.00	3.33	3.67	2.67	3.00	3.33	3.00	3.33	3.33	2.00	3.00	2.00	3.00	0.00	3.33	2.67	4.00	2.67	4.00	2.67	1.33	1.33	1.33	1.33	1.33	12.00	12.00	9.33	
stdev	1.00	0.58	0.58	0.58	1.00	1.53	0.00	1.15	0.58	1.00	0.00	1.00	0.00	1.15	1.53	1.00	0.58	1.00	0.58	1.53	1.15	1.53	1.15	1.15	1.15	3.61	1.00	1.53	
var	1.00	0.33	0.33	0.33	1.00	2.33	0.00	1.33	0.33	1.00	0.00	1.00	0.00	1.33	2.33	1.00	0.33	1.00	0.33	2.33	1.33	1.33	1.33	1.33	1.33	13.00	1.00	2.33	
skew	0.00	1.73	-1.73	-1.73	0.00	0.94	#DIV/0!	-1.73	1.73	0.00	#DIV/0!	-1.73	-0.94	0.00	-1.73	-0.94	0.00	-1.73	0.94	-1.73	0.94	-1.73	-1.73	-1.73	1.15	0.00	0.94		
COMPLETED and did not have CS 0																													
ln1	5	4	4	4		4	4	psu	4	4	wrd/psu	1	3	wrd	1	3	wrd	5		2		4		4		10	14	15	
DID NOT COMPLETE and did not have treatment CS 0																													
tc4						3	3	wrd/psu	3	3	mat	1	2	wrd	1	3	wrd									8	11		
DID NOT COMPLETE and did not have CS 0																													
ln2						1	1	mat/wrd	2	2	mat/wrd	1	1	wrd	3	4	pic/wrd										7	8	
ln3						3	3	wrd	2	2	mat	2	4	pic/wrd	0	1	wrd										7	10	
ln4						2	2	wrd/psu	2	3	psu	2	2	wrd	2	3	pic/wrd										8	10	
ln5						2	1	wrd	2	1	wrd	1	2	wrd	2	1	wrd										5	4	
count						4	4	4	3	3	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4	4
mean						2.00	1.75	2.00	2.33	1.50	2.25	1.50	2.25	1.25	2.25	1.25	2.25	1.25	2.25	1.25	2.25	1.25	2.25	1.25	1.25	6.75	8.00		
stdev						0.82	0.98	0.82	0.58	0.58	0.58	0.58	1.26	1.50	1.26	1.50	1.26	1.50	1.26	1.50	1.26	1.50	1.26	1.50	1.26	1.26	2.53		
var						0.67	0.92	0.67	0.33	0.33	0.33	0.33	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	1.58	8.00		
skew						0.00	0.85	#DIV/0!	1.73	0.00	1.13	-1.13	0.37													-1.13	-1.41		
DID NOT COMPLETE and had treatment CS 0																													
l26						3	4	psu	1	2	mat/psu	1	3	wrd	1	3	mat/wrd									6	12		
l27						3	4	psu	2	3	psu	1	2	wrd	3	3	pic									9	12		
l28						1	3	psu	4	3	psu	2	3	wrd	1	1	0	none								8	9		
count						3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3	3
mean						2.33	3.67	2.33	2.67	1.33	2.67	1.67	2.00													7.67	11.00		
stdev						1.15	0.58	1.53	0.58	0.58	0.58	1.15	1.73													1.53	1.73		
var						1.33	0.33	2.33	0.33	0.33	0.33	1.33	3.00													2.33	3.00		
skew						-1.73	-1.73	0.94	-1.73	1.73	-1.73	1.73	-1.73													-0.94	-1.73		

Appendix XII: BSU Data Mining Samples

UNSUP CLUSTER 7

NO selftraining
 NO POST TEST
~~39~~ 39

CLASS RESEMBLANCE STATISTICS

	Class 1	Class 2	Class 3	Class 4
Res. Score:	0.364	0.405	0.432	0.519
No. of Inst.	16	17	4	2
Cluster Quality:	0.08	0.21	0.29	0.54

INST SIM SCORE 37

to 1.0

DOMAIN STATISTICS FOR CATEGORICAL ATTRIBUTES

Number of Classes: 4
 Domain Res. Score: 0.34

Categorical Attribute Summary:	Name	Value	Frequency	Predictability
	group	c1	24	0.62
		t	15	0.38
preq1scr		3	17	0.44
		2	2	0.05
		5	5	0.13
		4	10	0.26
		1	4	0.10
		□	1	0.03
preq1sty		2	6	0.15
		4	22	0.56
		3	9	0.23
		5	1	0.03
		□	1	0.03
preq1met		psu	24	0.62
		wrd	7	0.18
		psu/wrd	3	0.08
		wrd/psu	1	0.03
		dia/wrd	1	0.03
		pic/wrd	1	0.03
		pic/psu	1	0.03
		□	1	0.03
		preq2scr		3
2	6			0.15
4	11			0.28
□	1			0.03
1	3			0.08
5	4			0.10
preq2sty		3	18	0.46
		4	13	0.33
		□	1	0.03
		2	5	0.13
		1	2	0.05
preq2met		mat	1	0.03
		psu	14	0.36
		wrd	4	0.10
		wrd/psu	6	0.15
		mat/psu	6	0.15
		□	1	0.03
		mat	4	0.10
		mat/wrd	2	0.05
wrd	1	0.03		
preq3scr		1	17	0.44
		3	4	0.10
		2	12	0.31
		□	5	0.13
		4	1	0.03
preq3sty		2	13	0.33
		3	12	0.31
		□	5	0.13
		1	7	0.18
		4	2	0.05

preq3met	wrd	28	0.72
	□	5	0.13
	pic/wrd	2	0.05
	psu	2	0.05
	wrd/psu	2	0.05
preq4scr	2	5	0.13
	4	9	0.23
	1	13	0.33
	3	6	0.15
	□	4	0.10
	0	1	0.03
	5	1	0.03
preq4sty	1	11	0.28
	4	2	0.05
	0	5	0.13
	2	6	0.15
	3	8	0.21
	□	4	0.10
	5	3	0.08
preq4met	wrd/pic	5	0.13
	pic/wrd	3	0.08
	none	5	0.13
	hpic	6	0.15
	pic	5	0.13
	wrd	8	0.21
	mat	1	0.03
	□	4	0.10
	mat/wrd	1	0.03
	pic	1	0.03
post1	4	7	0.18
	5	14	0.36
	2	5	0.13
	0	1	0.03
	3	7	0.18
	1	5	0.13
post2	4	6	0.15
	2	10	0.26
	1	12	0.31
	0	4	0.10
	3	4	0.10
	5	3	0.08
post3	2	10	0.26
	4	3	0.08
	1	16	0.41
	0	6	0.15
	3	4	0.10
post4	2	6	0.15
	0	12	0.31
	1	15	0.38
	4	1	0.03
	3	5	0.13
gender	M	23	0.59
	F	12	0.31
	□	4	0.10
year	4	7	0.18
	3	11	0.28
	2	17	0.44
	□	2	0.05
	1	2	0.05

age	over30	3	0.08
	18-22	28	0.72
	□	4	0.10
	23-29	4	0.10
race	C	24	0.62
	□	7	0.18
	NA	1	0.03
	Asian	7	0.18
HSgrade	b	9	0.23
	a	28	0.72
	□	1	0.03
	c	1	0.03
colgrade	b	16	0.41
	a	19	0.49
	c	3	0.08
	□	1	0.03
cs1309	c	9	0.23
	a	14	0.36
	b	12	0.31
	□	3	0.08
	d	1	0.03

DOMAIN STATISTICS FOR NUMERICAL ATTRIBUTES

	<u>Class 1</u>	<u>Class 2</u>	<u>Class 3</u>	<u>Class 4</u>	<u>Domain</u>	<u>Attribute Significance</u>
pretot1 (mean)	8.94	10.88	11.25	5.50	9.85	2.46
(sd)	1.77	2.15	1.26	2.12	2.33	
pretot2 (mean)	9.25	11.77	9.50	5.50	10.18	2.38
(sd)	2.11	2.14	2.38	2.12	2.64	
posttot (mean)	5.94	11.06	4.75	10.50	8.28	1.65
(sd)	2.65	3.17	1.50	2.12	3.83	
diff both (mean)	(3.16)	(0.27)	(5.63)	5.00	(1.73)	2.68
(sd)	2.29	3.99	2.87	4.24	3.96	

MOST COMMONLY OCCURRING CATEGORICAL ATTRIBUTE VALUES

	<u>Class 1</u>	<u>Class 2</u>	<u>Class 3</u>	<u>Class 4</u>
group	c1	t	c1	t
preq1scr	3	3	3	1
preq1sty	4	4	3	2
preq1met	psu	psu	wrd	wrd
preq2scr	3	3	4	1
preq2sty	3	3	3	1
preq2met	mat	psu	wrd/psu	wrd/psu
preq3scr	1	1	□	1
preq3sty	2	3	□	1
preq3met	wrd	wrd	□	wrd
preq4scr	1	4	3	1
preq4sty	1	3	2	0
preq4met	wrd	pic	hpic	none
post1	3	5	4	5
post2	1	4	1	2
post3	1	1	1	2
post4	1	2	0	3
gender	M	M	M	F
year	2	2	3	4
age	18-22	18-22	18-22	□
race	C	C	C	□
HSgrade	a	a	b	a
colgrade	b	a	c	a
cs1309	b	a	c	b

Class: 1
Total Number of Instances: 16
Class Resemblance Score: 0.36

Most Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
t	4	4	psu	2	3
c1	3	4	psu	5	4

Least Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
c1	4	4	psu	3	4
t	□	□	□	2	2

Categorical Attribute Summary:

<u>Name</u>	<u>Value</u>	<u>Frequency</u>	<u>Predictability</u>	<u>Predictiveness</u>
group	c1	12	0.75	0.50
	t	4	0.25	0.27
preq1scr	3	7	0.44	0.41
	2	2	0.13	1.00
	4	4	0.25	0.40
	5	1	0.06	0.20
	1	1	0.06	0.25
	□	1	0.06	1.00
preq1sty	2	4	0.25	0.67
	3	1	0.06	0.11
	4	10	0.63	0.45
	□	1	0.06	1.00
preq1met	psu	12	0.75	0.50
	wrd	2	0.13	0.29
	pic/wrd	1	0.06	1.00
	□	1	0.06	1.00
preq2scr	3	7	0.44	0.50
	4	2	0.13	0.18
	5	3	0.19	0.75
	1	2	0.13	0.67
	2	2	0.13	0.33
preq2sty	3	7	0.44	0.39
	2	3	0.19	0.60
	4	5	0.31	0.38
	1	1	0.06	0.50
preq2met	mat	1	0.06	1.00
	wrd	2	0.13	0.50
	mat	4	0.25	1.00
	psu	4	0.25	0.29
	wrd/psu	3	0.19	0.50
	mat/psu	1	0.06	0.17
	wrd	1	0.06	1.00
preq3scr	1	9	0.56	0.53
	2	5	0.31	0.42
	□	2	0.13	0.40
preq3sty	2	6	0.38	0.46
	3	5	0.31	0.42
	1	3	0.19	0.43
	□	2	0.13	0.40
preq3met	wrd	9	0.56	0.32
	pic/wrd	1	0.06	0.50
	psu	2	0.13	1.00
	□	2	0.13	0.40
	wrd/psu	2	0.13	1.00
preq4scr	2	2	0.13	0.40
	1	5	0.31	0.38
	4	3	0.19	0.33

	3	2	0.13	0.33
	☐	4	0.25	1.00
preq4sty	1	6	0.38	0.55
	0	2	0.13	0.40
	2	1	0.06	0.17
	☐	4	0.25	1.00
	3	2	0.13	0.25
	5	1	0.06	0.33
preq4met	wrd/pic	2	0.13	0.40
	none	2	0.13	0.40
	wrd	5	0.31	0.63
	hpic	1	0.06	0.17
	mat	1	0.06	1.00
	☐	4	0.25	1.00
	mat/wrd	1	0.06	1.00
post1	4	2	0.13	0.29
	2	4	0.25	0.80
	3	6	0.38	0.86
	5	1	0.06	0.07
	1	3	0.19	0.60
post2	4	1	0.06	0.17
	2	4	0.25	0.40
	1	8	0.50	0.67
	3	1	0.06	0.25
	0	2	0.13	0.50
post3	2	5	0.31	0.50
	0	3	0.19	0.50
	1	8	0.50	0.50
post4	2	1	0.06	0.17
	1	10	0.63	0.67
	0	5	0.31	0.42
gender	M	10	0.63	0.43
	☐	3	0.19	0.75
	F	3	0.19	0.25
year	4	4	0.25	0.57
	3	3	0.19	0.27
	2	8	0.50	0.47
	☐	1	0.06	0.50
age	over30	1	0.06	0.33
	18-22	9	0.56	0.32
	23-29	3	0.19	0.75
	☐	3	0.19	0.75
race	C	9	0.56	0.38
	☐	3	0.19	0.43
	NA	1	0.06	1.00
	Asian	3	0.19	0.43
HSgrade	b	3	0.19	0.33
	a	11	0.69	0.39
	☐	1	0.06	1.00
	c	1	0.06	1.00
colgrade	b	11	0.69	0.69
	a	4	0.25	0.21
	☐	1	0.06	1.00
cs1309	c	3	0.19	0.33
	b	8	0.50	0.67
	☐	2	0.13	0.67
	a	2	0.13	0.14

d 1 0.06 1.00

Attribute Values Necessary and Sufficient for Class Membership:

Name Value

Attribute Values Highly Sufficient for Class Membership:

Name Value

preq1scr 2
 preq1scr □
 preq1sty □
 preq1met pic/wrd
 preq1met □
 preq2met mat
 preq2met mat
 preq2met wrd
 preq3met psu
 preq3met wrd/psu
 preq4scr □
 preq4sty □
 preq4met mat
 preq4met □
 preq4met mat/wrd
 post1 2
 post1 3
 race NA
 HSgrade □
 HSgrade c
 colgrade □
 cs1309 d

Attribute Values Highly Necessary for Class Membership:

Name Value

Numerical Value Attribute Summary:	<u>Name</u>	<u>Mean</u>	<u>Standard Deviation</u>
	pretot1	8.938	1.769
	pretot2	9.25	2.113
	posttot	5.938	2.645
	diff both	-3.156	2.293

Class: 2
Total Number of Instances: 17
Class Resemblance Score: 0.41

Most Typical Instances:	<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
	t	3	4	psu	4	3
	t	5	4	psu	2	3

Least Typical Instances:	<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
	c1	4	4	psu	3	3
	t	3	4	pic/psu	4	3

Categorical Attribute Summary:	<u>Name</u>	<u>Value</u>	<u>Frequency</u>	<u>Predictability</u>	<u>Predictiveness</u>
	group	t	9	0.53	0.60
		c1	8	0.47	0.33
	preq1scr	3	8	0.47	0.47
		5	3	0.18	0.60
		4	5	0.29	0.50
		1	1	0.06	0.25
	preq1sty	4	11	0.65	0.50
		5	1	0.06	1.00
		3	4	0.24	0.44
		2	1	0.06	0.17
	preq1met	psu	12	0.71	0.50
		wrd	1	0.06	0.14
		psu/wrd	2	0.12	0.67
		dia/wrd	1	0.06	1.00

	pic/psu	1	0.06	1.00
preq2scr	2	4	0.24	0.67
	3	6	0.35	0.43
	□	1	0.06	1.00
	4	5	0.29	0.45
	5	1	0.06	0.25
preq2sty	4	7	0.41	0.54
	3	8	0.47	0.44
	□	1	0.06	1.00
	2	1	0.06	0.20
preq2met	psu	10	0.59	0.71
	wrd	1	0.06	0.25
	□	1	0.06	1.00
	mat/psu	4	0.24	0.67
	mat/wrd	1	0.06	0.50
preq3scr	3	3	0.18	0.75
	1	6	0.35	0.35
	4	1	0.06	1.00
	2	6	0.35	0.50
	□	1	0.06	0.20
preq3sty	3	7	0.41	0.58
	2	5	0.29	0.38
	1	3	0.18	0.43
	□	1	0.06	0.20
	4	1	0.06	0.50
preq3met	wrd	15	0.88	0.54
	□	1	0.06	0.20
	pic/wrd	1	0.06	0.50
preq4scr	4	6	0.35	0.67
	1	5	0.29	0.38
	0	1	0.06	1.00
	3	1	0.06	0.17
	2	3	0.18	0.60
	5	1	0.06	1.00
preq4sty	4	2	0.12	1.00
	3	6	0.35	0.75
	2	2	0.12	0.33
	1	5	0.29	0.45
	5	2	0.12	0.67
preq4met	pic/wrd	3	0.18	1.00
	pic	5	0.29	1.00
	hpic	2	0.12	0.33
	wrd	3	0.18	0.38
	pic	1	0.06	1.00
	wrd/pic	3	0.18	0.60
post1	5	10	0.59	0.71
	4	4	0.24	0.57
	1	2	0.12	0.40
	3	1	0.06	0.14
post2	2	3	0.18	0.30
	0	1	0.06	0.25
	3	3	0.18	0.75
	1	2	0.12	0.17
	4	5	0.29	0.83
	5	3	0.18	1.00
post3	4	3	0.18	1.00
	1	6	0.35	0.38
	3	4	0.24	1.00

	2	3	0.18	0.30
	0	1	0.06	0.17
post4	0	4	0.24	0.33
	4	1	0.06	1.00
	2	5	0.29	0.83
	3	4	0.24	0.80
	1	3	0.18	0.20
gender	F	7	0.41	0.58
	M	10	0.59	0.43
year	3	7	0.41	0.64
	2	8	0.47	0.47
	4	1	0.06	0.14
	1	1	0.06	0.50
age	18-22	15	0.88	0.54
	over30	1	0.06	0.33
	23-29	1	0.06	0.25
race	C	10	0.59	0.42
	□	3	0.18	0.43
	Asian	4	0.24	0.57
HSgrade	a	15	0.88	0.54
	b	2	0.12	0.22
colgrade	a	13	0.76	0.68
	b	4	0.24	0.25
cs1309	a	12	0.71	0.86
	□	1	0.06	0.33
	b	2	0.12	0.17
	c	2	0.12	0.22

Attribute Values Necessary and Sufficient for Class Membership:

Name Value

Attribute Values Highly Sufficient for Class Membership:

Name Value

preq1sty 5
 preq1met dia/wrd
 preq1met pic/psu
 preq2scr □
 preq2sty □
 preq2met □
 preq3scr 4
 preq4scr 0
 preq4scr 5
 preq4sty 4
 preq4met pic/wrd
 preq4met pic
 preq4met pic
 post2 4
 post2 5
 post3 4
 post3 3
 post4 4
 post4 2
 post4 3
 cs1309 a

Attribute Values Highly Necessary for Class Membership:

Name Value
 preq3met wrd
 age 18-22
 HSgrade a

Numerical Value Attribute Summary:		<u>Name</u>	<u>Mean</u>	<u>Standard Deviation</u>
		pretot1	10.882	2.147
		pretot2	11.765	2.137
		posttot	11.059	3.172
		diff both	-0.265	3.993

Class: 3
Total Number of Instances: 4
Class Resemblance Score: 0.43

Most Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
c1	3	3	wrd	4	3
c1	4	4	psu/wrd	4	3

Least Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
c1	5	3	wrd	3	3
c1	3	3	wrd	4	4

Categorical Attribute Summary:		<u>Name</u>	<u>Value</u>	<u>Frequency</u>	<u>Predictability</u>	<u>Predictiveness</u>
		group	c1	4	1.00	0.17
		preq1scr	3	2	0.50	0.12
			4	1	0.25	0.10
			5	1	0.25	0.20
		preq1sty	3	3	0.75	0.33
			4	1	0.25	0.05
		preq1met	wrd	3	0.75	0.43
			psu/wrd	1	0.25	0.33
		preq2scr	4	3	0.75	0.27
			3	1	0.25	0.07
		preq2sty	3	3	0.75	0.17
			4	1	0.25	0.08
		preq2met	wrd/psu	1	0.25	0.17
			mat/psu	1	0.25	0.17
			wrd	1	0.25	0.25
			mat/wrd	1	0.25	0.50
		preq3scr	□	2	0.50	0.40
			3	1	0.25	0.25
			2	1	0.25	0.08
		preq3sty	□	2	0.50	0.40
			4	1	0.25	0.50
			2	1	0.25	0.08
		preq3met	□	2	0.50	0.40
			wrd	2	0.50	0.07
		preq4scr	3	3	0.75	0.50
			1	1	0.25	0.08
		preq4sty	2	3	0.75	0.50
			0	1	0.25	0.20
		preq4met	hpic	3	0.75	0.50
			none	1	0.25	0.20
		post1	4	1	0.25	0.14
			0	1	0.25	1.00
			2	1	0.25	0.20
			5	1	0.25	0.07
		post2	1	2	0.50	0.17
			2	1	0.25	0.10

	0	1	0.25	0.25
post3	1	2	0.50	0.13
	0	2	0.50	0.33
post4	0	2	0.50	0.17
	1	2	0.50	0.13
gender	M	2	0.50	0.09
	□	1	0.25	0.25
	F	1	0.25	0.08
year	3	1	0.25	0.09
	4	1	0.25	0.14
	2	1	0.25	0.06
	□	1	0.25	0.50
age	18-22	3	0.75	0.11
	over30	1	0.25	0.33
race	C	4	1.00	0.17
HSgrade	b	4	1.00	0.44
colgrade	c	3	0.75	1.00
	b	1	0.25	0.06
cs1309	c	4	1.00	0.44

Attribute Values Necessary and Sufficient for Class Membership:

<u>Name</u>	<u>Value</u>
-------------	--------------

Attribute Values Highly Sufficient for Class Membership:

<u>Name</u>	<u>Value</u>
post1	0
colgrade	c

Attribute Values Highly Necessary for Class Membership:

<u>Name</u>	<u>Value</u>
group	c1
race	C
HSgrade	b
cs1309	c

Numerical Value Attribute Summary:

<u>Name</u>	<u>Mean</u>	<u>Standard Deviation</u>
pretot1	11.25	1.258
pretot2	9.5	2.38
posttot	4.75	1.5
diff both	-5.625	2.869

Class: 4
Total Number of Instances: 2
Class Resemblance Score: 0.52

Most Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
t	1	2	wrd	1	1
t	1	3	wrd/psu	4	2

Least Typical Instances:

<u>group</u>	<u>preq1scr</u>	<u>preq1sty</u>	<u>preq1met</u>	<u>preq2scr</u>	<u>preq2sty</u>
t	1	2	wrd	1	1
t	1	3	wrd/psu	4	2

Categorical Attribute Summary:

<u>Name</u>	<u>Value</u>	<u>Frequency</u>	<u>Predictability</u>	<u>Predictiveness</u>
group	t	2	1.00	0.13
preq1scr	1	2	1.00	0.50
preq1sty	2	1	0.50	0.17
	3	1	0.50	0.11

preq1met	wrd	1	0.50	0.14
	wrd/psu	1	0.50	1.00
preq2scr	1	1	0.50	0.33
	4	1	0.50	0.09
preq2sty	1	1	0.50	0.50
	2	1	0.50	0.20
preq2met	wrd/psu	2	1.00	0.33
preq3scr	1	2	1.00	0.12
preq3sty	1	1	0.50	0.14
	2	1	0.50	0.08
preq3met	wrd	2	1.00	0.07
preq4scr	1	2	1.00	0.15
preq4sty	0	2	1.00	0.40
preq4met	none	2	1.00	0.40
post1	5	2	1.00	0.14
post2	2	2	1.00	0.20
post3	2	2	1.00	0.20
post4	3	1	0.50	0.20
	0	1	0.50	0.08
gender	F	1	0.50	0.08
	M	1	0.50	0.04
year	4	1	0.50	0.14
	1	1	0.50	0.50
age	□	1	0.50	0.25
	18-22	1	0.50	0.04
race	□	1	0.50	0.14
	C	1	0.50	0.04
HSgrade	a	2	1.00	0.07
colgrade	a	2	1.00	0.11
cs1309	b	2	1.00	0.17

Attribute Values Necessary and Sufficient for Class Membership:

Name Value

Attribute Values Highly Sufficient for Class Membership:

Name Value
preq1met wrd/psu

Attribute Values Highly Necessary for Class Membership:

Name Value
group t
preq1scr 1
preq2met wrd/psu
preq3scr 1
preq3met wrd
preq4scr 1
preq4sty 0
preq4met none
post1 5

post2	2
post3	2
HSgrade	a
colgrade	a
cs1309	b

Numerical Value Attribute Summary:	<u>Name</u>	<u>Mean</u>	<u>Standard</u>
			<u>Deviation</u>
	pretot1	5.5	2.121
	pretot2	5.5	2.121
	posttot	10.5	2.121
	diff both	5	4.243

Rules for Class 1
16 instances

post4 = 1
:rule accuracy 66.67%
:rule coverage 62.50%

1.00 <= posttot <= 5.00
:rule accuracy 81.82%
:rule coverage 56.25%

colgrade = b
:rule accuracy 68.75%
:rule coverage 68.75%

**Total Percent Coverage = 100.00%

Rules for Class 2
17 instances

preq2met = psu
:rule accuracy 71.43%
:rule coverage 58.82%

post1 = 5
:rule accuracy 71.43%
:rule coverage 58.82%

12.00 <= pretot2 <= 16.00
:rule accuracy 75.00%
:rule coverage 52.94%

9.00 <= posttot <= 16.00
:rule accuracy 76.47%
:rule coverage 76.47%

-1.50 <= diff both <= 7.00

:rule accuracy 71.43%
:rule coverage 58.82%

colgrade = a

:rule accuracy 68.42%
:rule coverage 76.47%

cs1309 = a

:rule accuracy 85.71%
:rule coverage 70.59%

cs1309 = a

and 9.00 <= posttot <= 16.00

:rule accuracy 100.00%
:rule coverage 58.82%

****Total Percent Coverage = 100.00%**

Rules for Class 3

4 instances

colgrade = c

:rule accuracy 100.00%
:rule coverage 75.00%

****Total Percent Coverage = 75.00%**

Rules for Class 4

2 instances

2.00 <= diff both <= 2.00

:rule accuracy 66.67%
:rule coverage 100.00%

4.00 <= pretot2 <= 4.00
and 4.00 <= pretot1 <= 4.00
:rule accuracy 100.00%
:rule coverage 50.00%

****Total Percent Coverage = 100.00%**

SUPERVISED LEARN
11

using fit and
INDIVID SCORES.

Confusion Matrix

	Computed Class		
	c1	t	
c1	6	2	
t	1	4	

Percent Correct: 76.0%

Error: Upper Bound 47.7%

Error: Lower Bound sup

preq1scr	preq1sty	preq1met	preq2scr	preq2sty	preq2met	preq3scr	preq3sty	preq3met	preq4scr	preq4sty	preq4met	post1	post2	post3	post4	p4note	pretot1	pretot2	posttot	diff	group	computed	class
3	4	psu	3	3	mat	2	2	2	3	3	3	2	1	1	1	1	9	10	5	5	-4.5	c1	*
3	2	wrd	3	3	wrd/psu	3	2	2	3	3	3	3	1	1	0	0	8	7	5	5	-2.5	c1	*
4	4	psu	3	3	mat/psu	3	3	3	2	1	1	5	4	3	1	0	9	8	13	13	4.5	c1	*
5	4	psu	5	4	mat	3	3	3	3	3	3	3	1	1	0	0	10	8	5	5	-4	c1	*
1	2	pic/wrd	1	1	mat	1	1	1	4	5	5	2	1	0	0	1	7	9	4	4	-4	c1	*
3	4	psu	5	4	wrd/psu	1	1	1	1	0	0	1	2	0	0	0	10	9	5	5	-4.5	c1	*
3	4	psu	5	4	psu	1	1	1	1	0	0	4	3	2	2	0	10	12	11	11	0	c1	*
5	4	psu	2	3	mat/wrd	2	2	2	5	3	5	4	4	2	2	1	14	14	12	12	-2	t	*
3	4	pic/psu	4	3	mat/wrd	3	4	4	4	5	5	5	5	5	0	2	14	16	12	12	-3	t	*
4	2	psu/wrd	4	4	psu	2	3	3	1	1	1	4	2	3	0	0	11	10	9	9	-1.5	c1	*
4	4	psu	2	3	mat/psu	1	3	3	2	1	3	3	1	0	1	0	7	10	5	5	-3.5	c1	*
3	3	psu	2	2	wrd	1	2	2	2	1	1	1	0	0	0	0	5	5	1	1	-4	t	*
3	3	psu	3	4	psu	1	1	1	1	2	2	5	4	4	4	3	8	10	16	16	7	t	*

CLASS RESEMBLANCE STATISTICS

	<u>Class c1</u>	<u>Class t</u>	<u>Domain</u>
Res. Score:	0.285	0.287	0.28
No. of Inst.	16	10	26
Class Significance:	0.03	0.04	

DOMAIN STATISTICS FOR CATEGORICAL ATTRIBUTES

Number of Classes:	2
Domain Res. Score:	0.28

<u>Categorical Attribute Summary:</u>	<u>Name</u>	<u>Value</u>	<u>Frequency</u>	<u>Predictability</u>
	group	c1	16	0.62
		t	10	0.38
	preq1scr	3	11	0.42
		2	2	0.08
		5	3	0.12
		4	7	0.27
		1	3	0.12
	preq1sty	2	3	0.12
		3	8	0.31
		4	14	0.54
		5	1	0.04
	preq1met	psu	16	0.62
		wrd	6	0.23
		psu/wrd	2	0.08
		wrd/psu	1	0.04
		dia/wrd	1	0.04
	preq2scr	3	10	0.38
		4	9	0.35
		□	1	0.04
		2	3	0.12
		1	2	0.08
		5	1	0.04
	preq2sty	3	12	0.46
		□	1	0.04
		4	8	0.31
		2	4	0.15
		1	1	0.04
	preq2met	mat	1	0.04
		wrd	4	0.15
		wrd/psu	4	0.15
		mat/psu	3	0.12
		□	1	0.04
		psu	11	0.42
		mat	1	0.04
		mat/wrd	1	0.04

preq3scr	1	11	0.42
	2	9	0.35
	□	2	0.08
	3	3	0.12
	4	1	0.04
preq3sty	2	10	0.38
	3	10	0.38
	□	2	0.08
	1	3	0.12
	4	1	0.04
preq3met	wrd	21	0.81
	□	2	0.08
	pic/wrd	1	0.04
	psu	2	0.08
preq4scr	2	3	0.12
	1	9	0.35
	3	5	0.19
	4	7	0.27
	□	1	0.04
	0	1	0.04
preq4sty	1	8	0.31
	0	4	0.15
	2	5	0.19
	3	6	0.23
	4	2	0.08
	□	1	0.04
preq4met	wrd/pic	2	0.08
	none	4	0.15
	hpic	5	0.19
	pic	4	0.15
	pic/wrd	2	0.08
	wrd	5	0.19
	mat	1	0.04
	□	1	0.04
	mat/wrd	1	0.04
pic	1	0.04	
post1	4	5	0.19
	2	3	0.12
	5	10	0.38
	0	1	0.04
	3	4	0.15
	1	3	0.12
p1note	□	24	0.92
	notcode	1	0.04
	pic	1	0.04

post2	4	3	0.12
	2	8	0.31
	1	7	0.27
	0	3	0.12
	3	3	0.12
	5	2	0.08
p2note	□	22	0.85
	notcode	2	0.08
	wrd	1	0.04
	psu	1	0.04
post3	2	7	0.27
	1	12	0.46
	0	3	0.12
	3	2	0.08
	4	2	0.08
p3note	□	23	0.88
	pic	3	0.12
post4	2	4	0.15
	1	11	0.42
	0	6	0.23
	4	1	0.04
	3	4	0.15
p4note	pic	1	0.04
	□	20	0.77
	notcode	1	0.04
	wrd	4	0.15

DOMAIN STATISTICS FOR NUMERICAL ATTRIBUTES

	<u>Class c1</u>	<u>Class t</u>	<u>Domain</u>	<u>Attribute Significance</u>
pretot1 (mean)	10.19	9.90	10.08	0.13
(sd)	1.64	3.00	2.21	
pretot2 (mean)	10.06	10.80	10.35	0.29
(sd)	1.98	3.33	2.55	
posttot (mean)	7.63	9.80	8.46	0.62
(sd)	3.81	2.70	3.54	
diff both (mean)	(2.50)	(0.55)	(1.75)	0.46
(sd)	4.31	3.96	4.21	

MOST COMMONLY OCCURRING CATEGORICAL ATTRIBUTE VALUES

	<u>Class c1</u>	<u>Class t</u>
group	c1	t
preq1scr	3	3
preq1sty	4	4
preq1met	psu	psu
preq2scr	3	4
preq2sty	3	4
preq2met	psu	psu

preq3scr	1	1
preq3sty	3	2
preq3met	wrd	wrd
preq4scr	1	4
preq4sty	1	3
preq4met	hpic	wrd
post1	5	5
p1note	<input type="checkbox"/>	<input type="checkbox"/>
post2	1	2
p2note	<input type="checkbox"/>	<input type="checkbox"/>
post3	1	2
p3note	<input type="checkbox"/>	<input type="checkbox"/>
post4	1	2
p4note	<input type="checkbox"/>	<input type="checkbox"/>

Rules for Class c1
16 instances

preq2sty = 3
:rule accuracy 83.33%
:rule coverage 62.50%

post4 = 1
:rule accuracy 81.82%
:rule coverage 56.25%

10.00 <= pretot1 <= 13.00
:rule accuracy 76.92%
:rule coverage 62.50%

10.00 <= pretot2 <= 13.00
:rule accuracy 71.43%
:rule coverage 62.50%

3.00 <= posttot <= 10.00
:rule accuracy 72.22%
:rule coverage 81.25%

-9.00 <= diff both <= -2.00
:rule accuracy 73.33%
:rule coverage 68.75%

**Total Percent Coverage = 100.00%

Rules for Class t
10 instances

-2.50 <= diff both <= 2.00
:rule accuracy 70.00%
:rule coverage 70.00%